(continued from part 14)

Structured programming

As we have said, computer programs should be developed according to the top-down design approach, i.e. in a well-structured, modular fashion. This method divides the program into modules (sub-programs or sub-routines) which are separately written and tested to form small, well-defined portions of the final program. Such is the relationship between top-down design and the quality of the resulting program, that the concept of structured programming can be summarised by listing the characteristics of a 'good' program as follows:

1) the problem is specified in a complete and clear way;

2) development is in a modular and orderly manner with concise readable documentation;

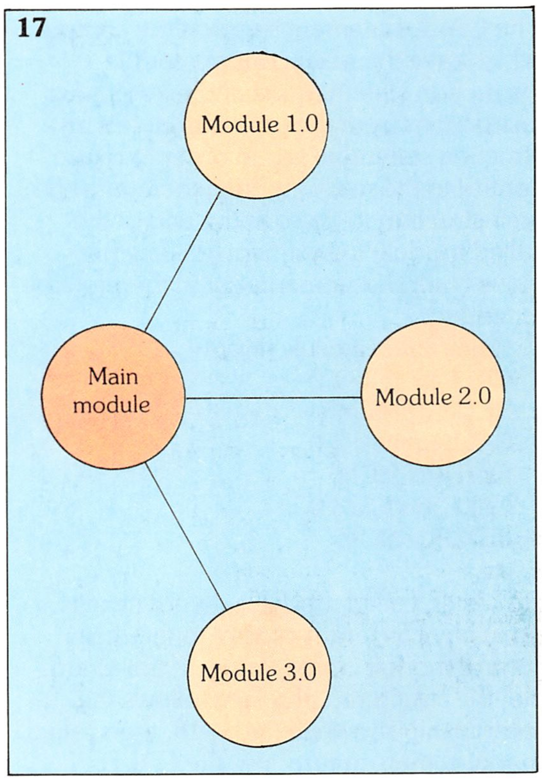
3) modules are small with clearly defined purposes.

A structured program is written so that a **main control module** assumes control over all other modules, calling for them only when required (*figure 17*). Once a called module has been run, control returns to the main module. In the example shown in *figure 17*, a list of control statements that would enable the main control module to call other modules could be written as follows:

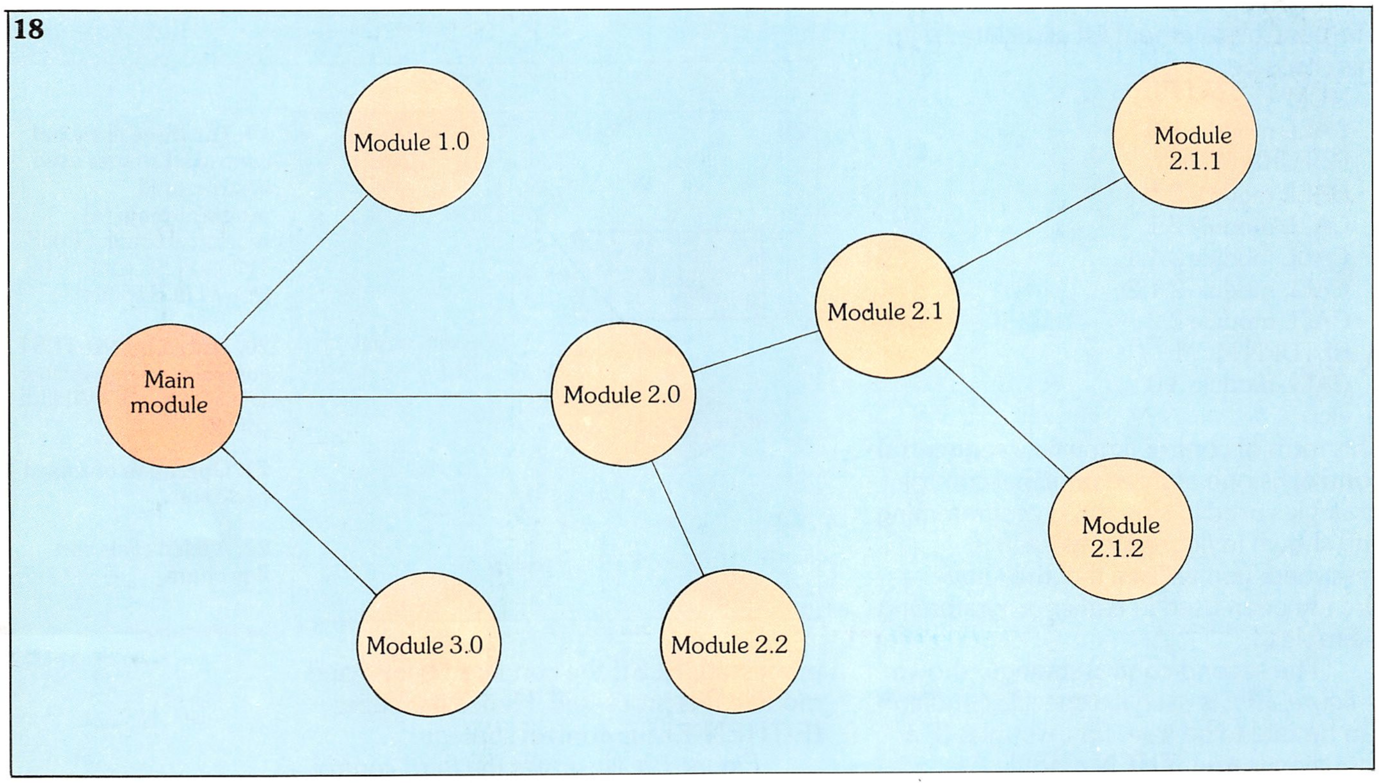
17. Elements of a structured program – the main control module calls other modules when required.

18. A structured program where the modules called by the main module, in turn call other modules from a lower level.

17



18



```
Main control module
CALL module 1.0
Main control module
CALL module 2.0
Main control module
etc.
```

The CALL statement instructs the computer to leave the main control module instruction statement list and take its next instructions from the called module's instruction statement list. In order to return control to the main module, there must be an instruction to do so at the end of the called module's statement list. The previous control statements could therefore be extended:

```
Main control module (MCM)
CALL module 1.0
RETURN MCM
CALL module 2.0
RETURN MCM
CALL module 3.0
RETURN MCM
etc.
```

However, called modules do not necessarily have to return control to the main control module immediately. If, for example, the structured program follows the relationship shown in *figure 18*, then called modules may, in turn, use the CALL statement to call modules of a lower level. The control statement list associated with this could be:

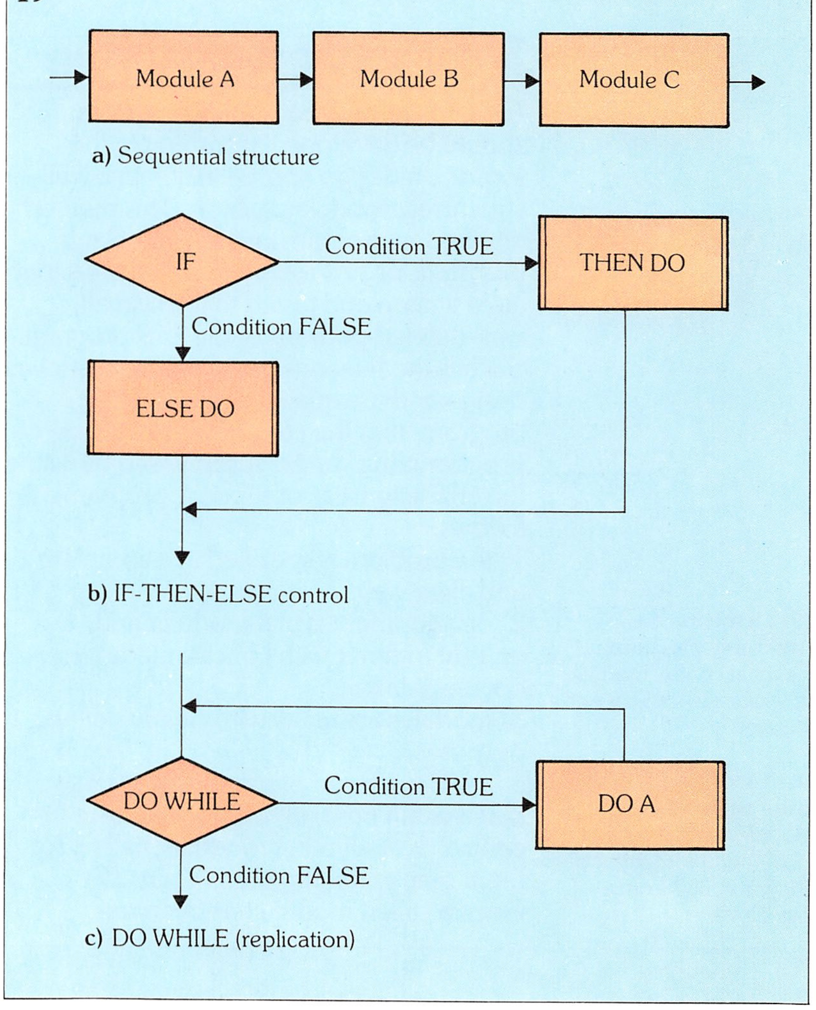
```
MCM
CALL module 1.0
RETURN MCM
CALL module 2.0
CALL module 2.1
CALL module 2.1.1
CALL module 2.1.2
CALL module 2.2
RETURN MCM
CALL module 3.0
etc.
```

This form of control, known as **sequential control**, is one of three principal control strategies used in structured programming and shown in *figure 19*. As each is described, you will see that they have already been used in earlier programming examples.

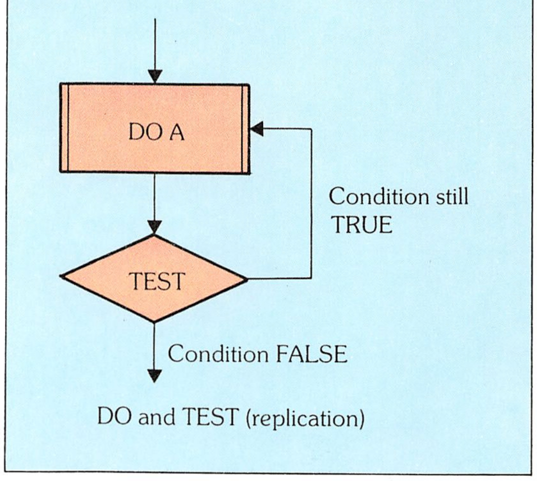
The second control strategy, shown in *figure 19b*, is used if a specific condition can be said to be true, for example: **IF** a condition is true, **THEN** module A is processed, **ELSE** the condition is false and module B is processed. Hence the name **IF-THEN-ELSE control** strategy.

Figure 19c illustrates the third control

19



20



19. The three principal control strategies used in structured programming: (a) sequential control; **(b)** IF-THEN-ELSE control; **(c)** DO-WHILE control.

20. The DO-AND-TEST control strategy which is similar to the DO-WHILE control.

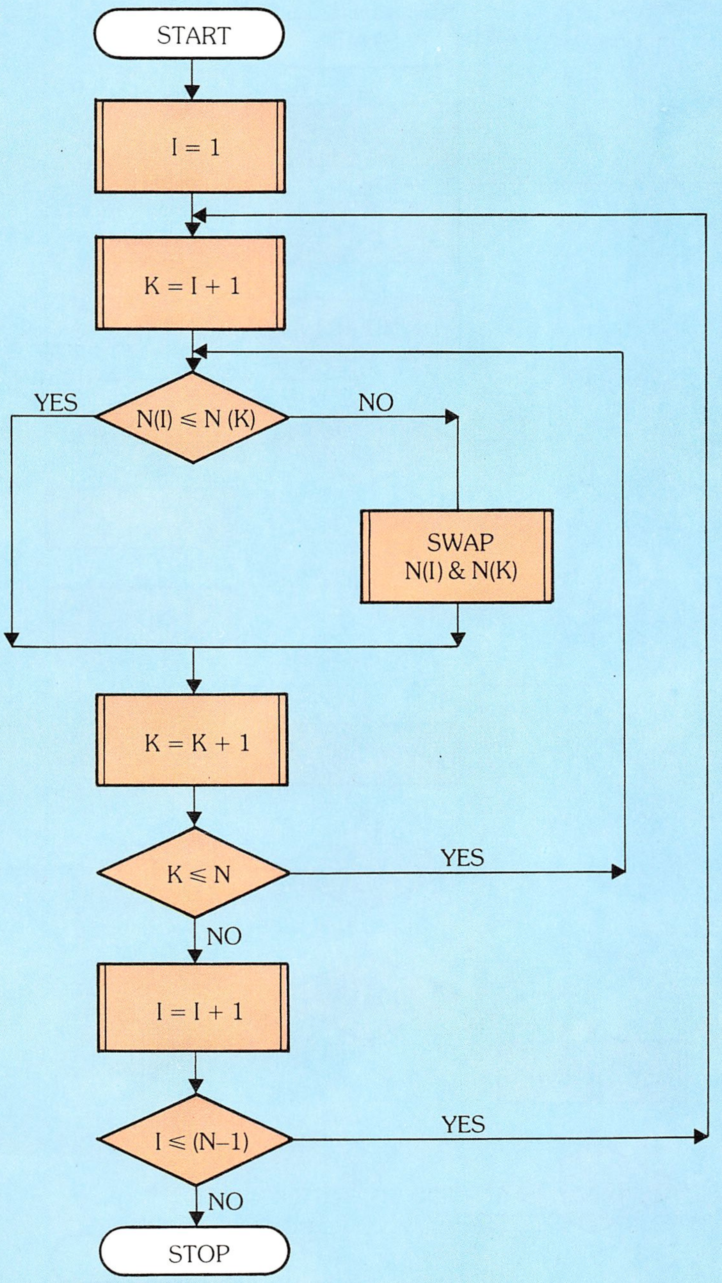
21. Operation of a fixed cycle sort.

22. A fixed cycle sort flowchart.

21

	Sort cycles									List
Start	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**position**
135	3	3	3	3	3	3	3	3	3	1
40	135	14	14	14	14	14	14	14	14	2
3	40	135	40	40	40	40	40	40	40	3
80	80	80	135	65	65	65	65	65	65	4
127	127	127	127	135	80	80	80	80	80	5
65	65	65	80	127	135	127	127	127	127	6
830	830	830	830	830	830	830	135	135	135	7
415	415	415	415	415	415	415	830	250	250	8
250	250	250	250	250	250	250	415	830	415	9
14	14	40	65	80	127	135	250	415	830	10

22



strategy which is used when an operation needs to be repeated. This **DO-WHILE control** instructs the computer to **DO** a sequence of instructions **WHILE** a particular condition is true. **Loops** or iterations are possible within a computer program with this type of control.

An experienced computer programmer should be able to develop any program using only these three program strategies. However, adaptations do exist, an example of one is shown in *figure 20* – the **DO AND TEST control** strategy. This is similar to the DO WHILE strategy, the difference being when the condition is checked. In DO WHILE control the condition for executing module A is first checked and, if found to be not true, module A is not executed. A control statement list for this could be (where I and J are two variables):

```
WHILE I < J, CALL A (i.e. DO A)
RETURN              (i.e. end A)
WHILE I < J, CALL A
etc.
```

In DO AND TEST control, module A is first processed, then the condition is tested. A control statement list could be:

```
CALL A
TEST I < J
CALL A
etc.
```

Applications of structured programs

A **sorting program** is one which can order an unordered list of data items according to the various criteria of size, numerical value, alphabetical sequence etc. They are often used as sub-routines in larger programs and are an extremely valuable programming tool.

The two most common sorting programs are the **fixed cycle sort** and the **bubble sort**. For our examples we shall use a list of ten random numbers and sort them into ascending order, i.e. with the lowest number at the beginning of the list.

The ten unordered numbers are stored in the array of locations:

N(1), N(2), N(3) ....N(10)

which can be known as N(I), where I varies from 1 to 10 and where:

N(I) = 135, 40, 3, 80, 127, 65, 830, 415, 250, 14

i.e. the 10 unordered numbers.

All three sort programs have a **sort cycle** where one number is compared with another. If the first number is greater than the second, then the numbers are transposed or swapped over; if the second is greater than the first, they are not transposed. The difference between the three programs lies mainly in the use of this sort cycle.

Fixed cycle sort

The fixed cycle sort uses a fixed number of sort cycles – hence the name. If N variables are to be sorted, then there will be a total of N-1 sort cycles. In every cycle, a comparison is made between a number and all those after it in the list, transposing the numbers if they are not in order.

In the first cycle, the first number in the list, stored in variable N(1), is compared with the second number, stored in variable N(2); if N(1) is greater than N(2) the numbers are swapped. Variable N(1) is then compared with variable N(3), and so on, to the end of the cycle. The second cycle compares the second variable, N(2) with variables N(3), N(4), N(5) etc. and so the number of comparisons in successive cycles is therefore reduced by one each time.

The operation of the fixed cyclic sort is shown in the table of *figure 21*. After the first cycle, the number 3 (the lowest number in the list) is in position 1; after the second cycle, the number 14 (the next lowest number) is in position 2; and so on.

Figure 22 shows a fixed cycle sort flowchart. Variable I is used as a pointer, and initially points to the first position in the array. It is incremented by one at the end of every cycle until the N-1 cycle has been carried out. Variable K, on the other hand, is used as a counter starting at I + 1 every cycle, and reaches N.

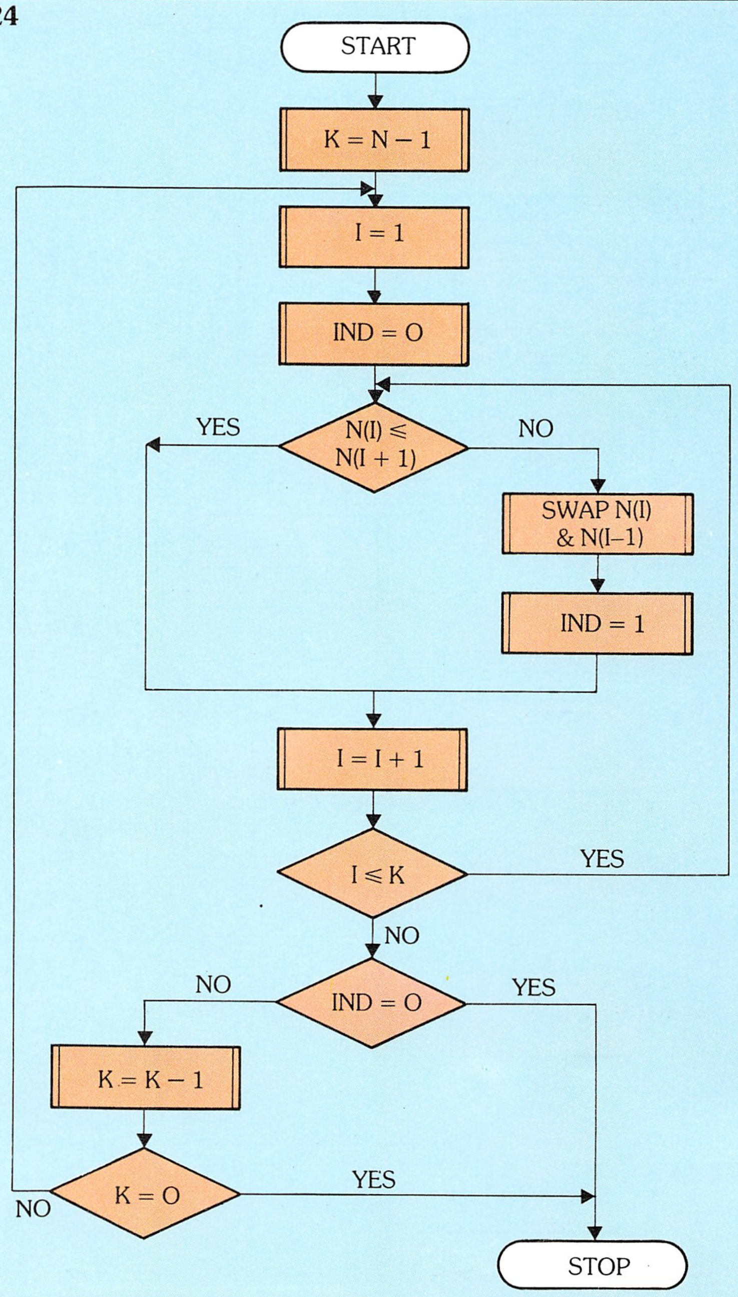
The flowchart clearly shows the external loop controlled by I that is repeated N-1 times, and the internal loop controlled by K that is repeated N-1 times to begin with, down to only once in the final loop.

Although this sort program is easy to understand and simple to program, it is unnecessarily long. Even if the array of numbers was perfectly sorted before the program was applied, *all the cycles and loops are still carried out.*

23

	Sort cycles									
Start	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**List position**
135	40	3	3	3	3	3	3	3	3	1
40	3	40	40	40	40	40	40	14	14	2
3	80	80	65	65	65	65	14	40	40	3
80	127	65	80	80	80	14	65	65	65	4
127	65	127	127	127	14	80	80	80	80	5
65	135	135	135	14	127	127	127	127	127	6
830	415	250	14	135	135	135	135	135	135	7
415	250	14	250	250	250	250	250	250	250	8
250	14	415	415	415	415	415	415	415	415	9
14	830	830	830	830	830	830	830	830	830	10

24



23. Operation of a bubble sort – note that the list is in order after the eighth cycle.

24. Flowchart for a bubble sort.

Bubble sort

The bubble sort is a more efficient method of sorting where, although the maximum number of sort cycles is N-1, the *actual* number of cycles depends on the condition of the numbers to be sorted; only adjacent numbers are compared (and transposed if required), the first with the second, the second with the third, the third with the fourth and so on. At the start of every cycle, a flag is reset which becomes set when a swap is made. If the flag is still reset at the end of a sort cycle then the list is in order. This is an obvious improvement over the fixed cycle sort because if the list of numbers was already in order then only one cycle would be carried out.

At the end of the first sort cycle of a bubble sort, the greatest number lies at the bottom of the list while the lower numbers move towards the top like bubbles rising in water, hence the name. Every sort cycle requires one less comparison than the previous sort cycle as with the fixed cycle sort. As can be seen in *figure 23* the list is in order after the eighth cycle, but in our example an extra cycle is required to leave the swap flag in a reset condition. Other lists could be sorted in fewer cycles.

A flowchart for a bubble sort program is shown in *figure 24*. In this case it is impossible to estimate how many instructions will be carried out, because the number of sort cycles is variable.

Glossary

bubble sort	sort program in which the smallest variables in an array bubble to the top, i.e. to the first position in the array
DO AND TEST	a program strategy which implements a module, then tests a condition. If the condition is not met the module is implemented again
DO WHILE	a program strategy which checks a condition then implements a module if required
fixed cycle sort	a sort program which has a fixed number of sort cycles, regardless of the order of the variables
IF-THEN-ELSE	a program strategy in which a condition is checked and the outcome defines which module is then implemented
main control module	module in a structured program which assumes control over all other modules, calling for other modules only when required
sequential control	a program strategy in which each module is processed sequentially e.g. module A, module B, module C
sort	the process of putting numeric or alphabetic data in ascending or descending order
structured programming	development of computer programs such that they are built up in a logical modular fashion

ELECTRICAL TECHNOLOGY

Inductors and self inductance

In the previous *Basic Theory Refresher* on electomagnetic induction we saw that an EMF is generated in a single turn of wire when the flux that is linking with it changes. We have also found that the magnitude of the generated voltage is equal to the rate of change of flux linking with the turn of wire. If the flux in *figure 1*, linking with a single turn of wire, changes from ϕ_1 to ϕ_2 in a period of time T, then the voltage generated, E_1, will be given by:

$$E_1 = -\frac{\phi_2 - \phi_1}{T}$$

If we now wind the wire into a coil of N turns (*figure 2*), there will be a voltage, E, generated in each turn of the coil. Since there are N turns of wire in series, the total voltage is given by the sum of the voltages generated in each turn. The total voltage in the coil is therefore:

$$\begin{aligned} E &= N \times E_1 \\ &= -N\frac{\phi_2 - \phi_1}{T} \\ &= -N\frac{\phi_{L2} - \phi_{L1}}{T} \end{aligned}$$

where ϕ_{L1} is the total number of initial flux linkages (i.e. $\phi_{L1} = N\phi_1$). Similarly ϕ_{L2} is the total number of flux linkages after time T.

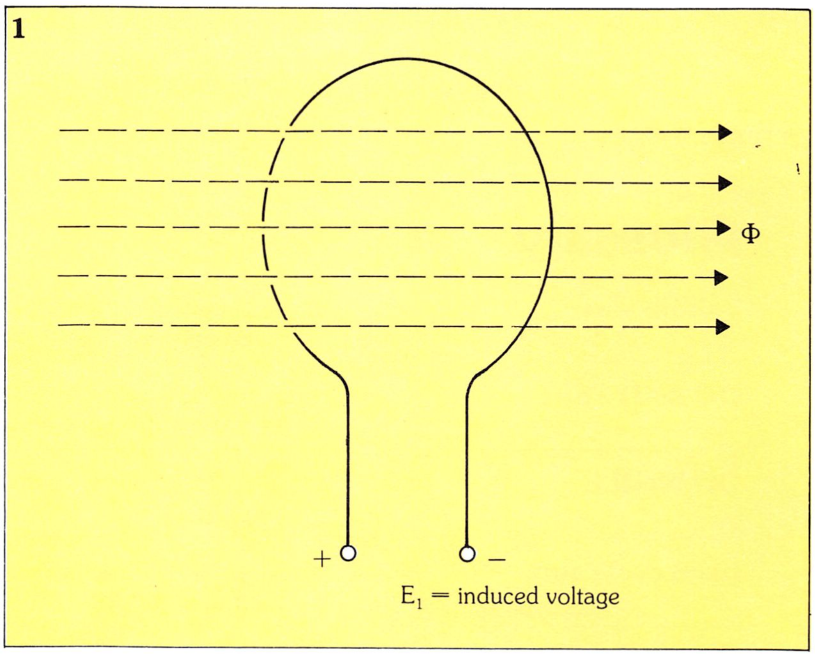
Let's look at the way that this flux is being created. If a current, I, is passed through the coil shown in *figure 2*, then a magnetic field of strength H will be set up, where:

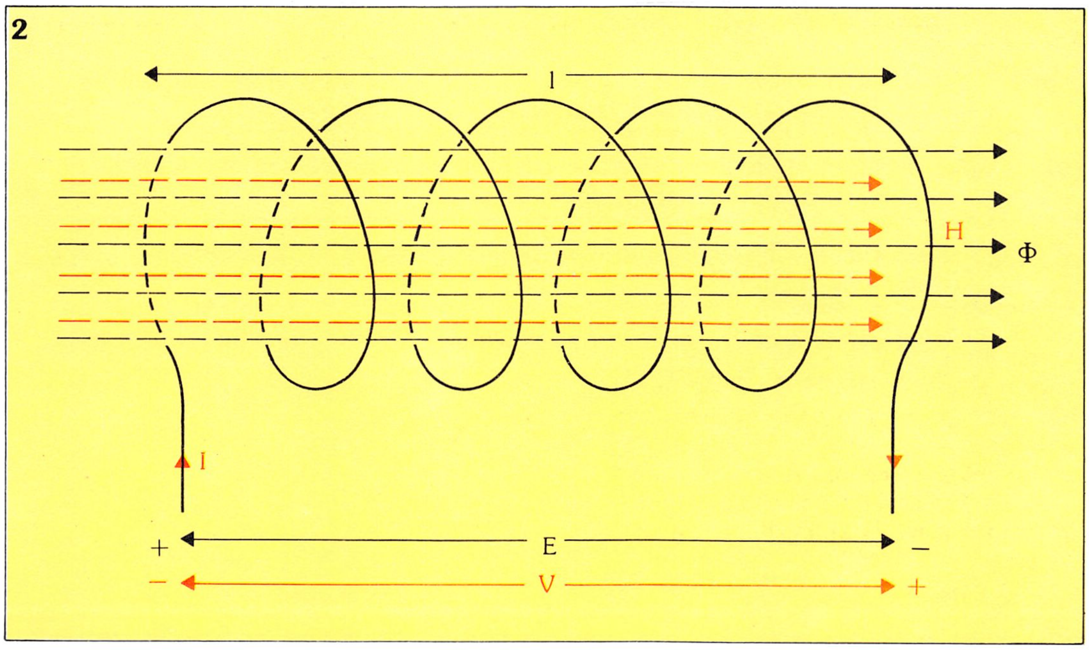
$$H = \frac{NI}{l}$$

If we consider this coil to be wound around a former of relative permeability μ_r, the flux density, B, will be:

$$\begin{aligned} B &= \mu_o \mu_r H \\ &= \frac{\mu_o \mu_r NI}{l} \end{aligned}$$

The total flux, ϕ, across the cross-sectional





1. Flux linkages with a single turn of wire generating a voltage, E_1.

2. N turns of wire in series generates a voltage, E, which is the sum of the voltages generated in each turn.

area, A, of the coil is then:

$$= BA$$

$$= \frac{\mu_o \mu_r NAI}{l}$$

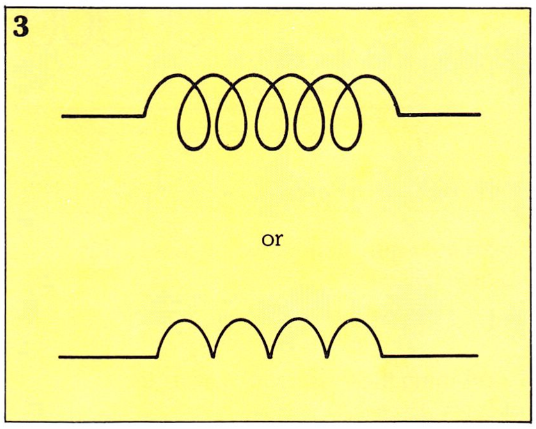
As this flux links with each turn of the coil, the total flux linkage, ϕ_L, is:

$$\phi_L = N\phi$$

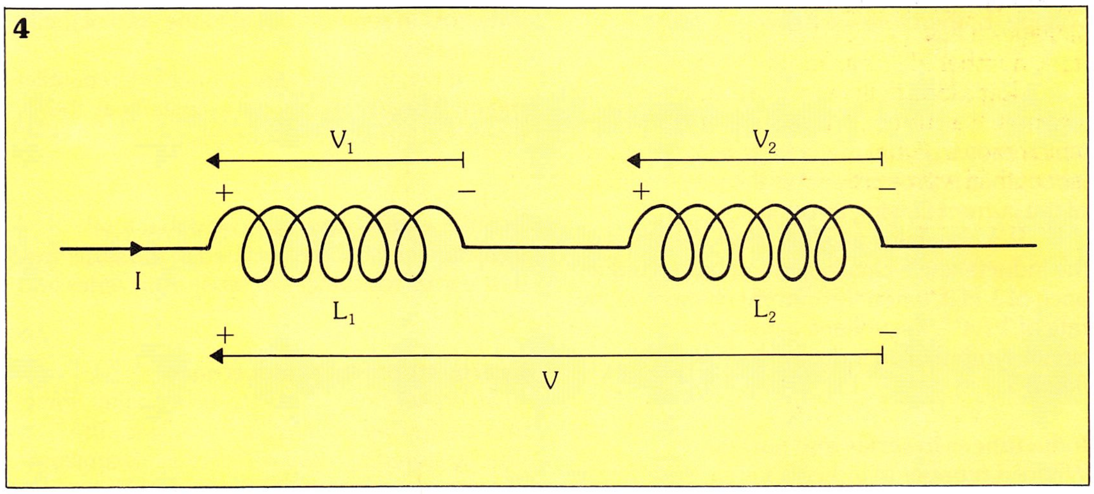
$$= \frac{\mu_o \mu_r N^2AI}{l}$$

Self inductance

In general terms, we can see that the flux linked is proportional to the amount of current flowing and this can be written as:



3. The circuit symbol for an inductor



4. Two inductors connected in series.

$$\phi_L = LI$$

where L is the self inductance of the circuit. The circuit element that possesses this property is known as an **inductor**. Self inductance is measured in henrys (symbol H); a circuit has a self inductance of 1 H, if a flux of 1 Wb is created by a current of 1 A flowing through it.

Let's look at the unit of the henry in a little more detail. Rewriting the equation above gives us:

$$L = \frac{\phi_L}{I}$$

Substituting the names of the units for their symbols on both sides of this equation gives us:

$$\text{henry} = \frac{\text{weber}}{\text{amp}}$$

We have previously seen that the volt-second is an alternative expression for the weber, so:

$$\text{henry} = \frac{\text{volt-second}}{\text{amp}}$$

Also from Ohm's law we know that:

$$\text{amp} = \frac{\text{volt}}{\text{ohm}}$$

This finally gives us:

$$\text{henry} = \text{ohm-second}$$

that is,

$$H = \Omega s$$

From the previous equations, we can also see that the inductance of a long straight solenoid is given by:

$$L = \frac{\mu_o \mu_r N^2 A}{l}$$

This expression is also true for an inductor wound onto a toroidal former, if we take l to be the toroid's mean circumference.

Inductance can be looked at in another way. We know that the reluctance, S, is related to the flux, ϕ, and magnetomotive force, F, by:

$$F = S\phi$$

Since:

$$F = NI$$

and:

$$\phi_L = N\phi$$

we obtain:

$$\phi_L = \frac{NF}{S}$$

$$= \frac{N^2I}{S}$$

inductance can therefore be written as:

$$L = \frac{N^2}{S}$$

This equation shows us that inductance is a property only of the number of turns of wire on the coil, and the reluctance of the former onto which it is wound. Inductance is independent of the current flowing in the coil if the coil is wound in air and only dependent on the current for an iron cored inductor to the extent that the permeability of iron is non-linear.

Effect of an inductance in a circuit

We can now find out what happens when the current in the circuit changes. Assuming that the current increases from a value of I_1 to I_2 in time, T, the resulting flux will rise from:

$$\phi_1 = LI_1$$

to:

$$\phi_2 = LI_2$$

We know that the changing flux will give rise to an induced voltage, E, where:

$$E = -\frac{\phi_2 - \phi_1}{T}$$

$$= -L\frac{I_2 - I_1}{T}$$

In order to maintain this current flow an external voltage, V, is applied to the terminals of the coil. This external voltage has to be exactly equal and opposite to the induced voltage, E, therefore we have:

$$V = -E$$

$$= L\frac{I_2 - I_1}{T}$$

which shows us that the voltage across any inductance is proportional to the rate of change of the current flowing through it.

We now have an alternative definition of the inductance: a circuit element has an inductance of 1 H if, when the current changes at the rate of 1 As^{-1}, the voltage across it is 1 V. The circuit symbol of an inductor is shown in *figure 3*.

Inductances in series and parellel

Where a number of inductors are connected in series (*figure 4*) we can find the value of the single equivalent inductor. When the current through the circuit changes from I_1 to I_2 in time T, the voltage, V, across inductor L_1, will be:

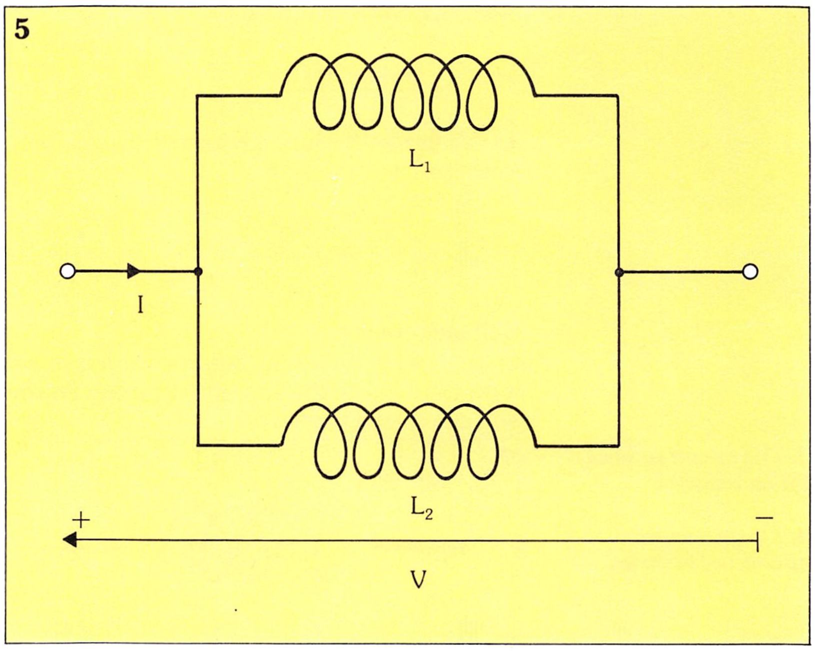
$$V_1 = L_1\frac{I_2 - I_1}{T}$$

Similarly, voltage V_2 across unductor L_2 will be given by:

$$V_2 = L_2\frac{I_2 - I_1}{T}$$

The total voltage, V, across both inductors is:

$$V = V_1 + V_2$$



5. Two inductors connected in parallel.

and if the equivalent inductance is L, this voltage will be:

$$V = L\frac{I_2 - I_1}{T}$$

Combining these equations gives us:

$$L = L_1 + L_2$$

The total inductance, therefore, of a number of inductors in series, is equal to the sum of their individual values.

If two inductors are connected in parallel (*figure 5*), we can prove in a similar way that their total inductance, L, is given by:

$$\frac{1}{L} = \frac{1}{L_1} + \frac{1}{L_2}$$

In other words, we have found that total inductances can be calculated in the same way as the total resistances of resistors in series and parallel.

Energy stored in an inductance

In an earlier *Basic Theory* article on flux, force and energy we obtained a formula for the energy stored in a capacitor. We can similarly obtain an expression for the energy W (in joules) stored in an inductor of L henrys, carrying a current of I amps:

$$W = \tfrac{1}{2}LI^2$$

Review

We have established here, that whenever the current through an inductor is changing, a voltage must be applied across the inductor to maintain this changing current. If the voltage falls to zero, then the current flowing in the inductor will remain constant. □

ELECTRICAL TECHNOLOGY

Mutual inductance

In the *Basic Theory Refresher* on self inductance we saw how a flux is created by a current through a coil and how this flux links with the coil which created it. This concept of **self inductance** can be extended as shown in *figure 1* where a current, I_1, flowing in the left-hand coil (coil 1) creates a flux, ϕ_1, which links with the right-hand coil (coil 2).

There are N turns in coil 2 so the total number of flux linkages is given by:

$$\phi_{L2} = N\phi_1$$

and, since this is directly related to the magnitude of the current through coil 1, we write:

$$\phi_{L2} = MI_1$$

where M is a constant.

Mutually induced voltage

If the flux of the primary coil, (say, ϕ_1 of *figure 1*) is varying, a voltage E_2 will be generated in the secondary coil (coil 2 of *figure 1*), the magnitude of which is given by the expression:

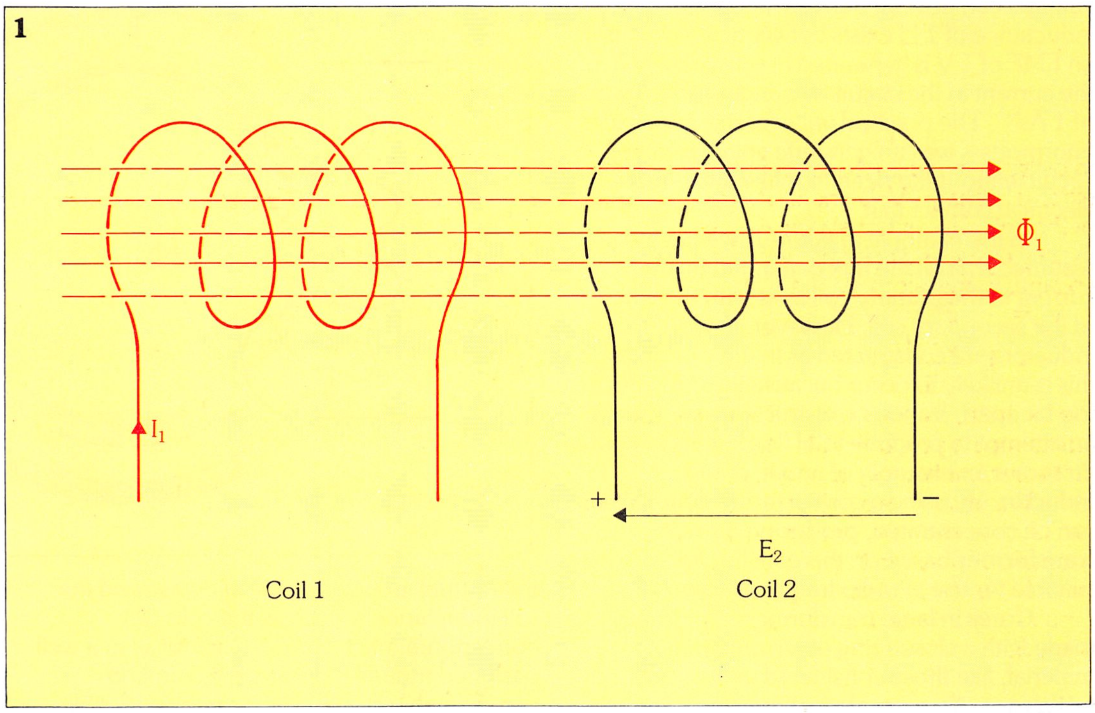
$$E_2 = -\frac{\phi''_{L2} - \phi'_{L2}}{T}$$

where ϕ'_{L2} and ϕ''_{L2} are the initial and final values of the flux linked with the secondary coil during time T.

The flux values ϕ'_{L2} and ϕ''_{L2} must be created by current I'_1 and I''_1 so can be included in the above expression, to give:

$$E_2 = -M\frac{I''_1 - I'_1}{T}$$

1. A changing flux in a primary coil (coil 1) induces a voltage (E_2) in a secondary coil (coil 2).



Similarly, a current I_2 flowing through the right-hand coil, as in *figure 2*, will create a flux, ϕ_2, which links with the left-hand coil giving a flux linkage ϕ_{L1}. This linkage is related to the current I_2, such that:

$$\phi_{L1} = MI_2$$

where the constant M is *the same as in the previous expression*, and is known as the **mutual inductance**.

The unit of mutual inductance, like that of self inductance, is the henry and can be defined as follows: a pair of coils have a mutual inductance of 1 H if a current of 1 A in one coil produces a flux linkage of 1 Wb in the other coil.

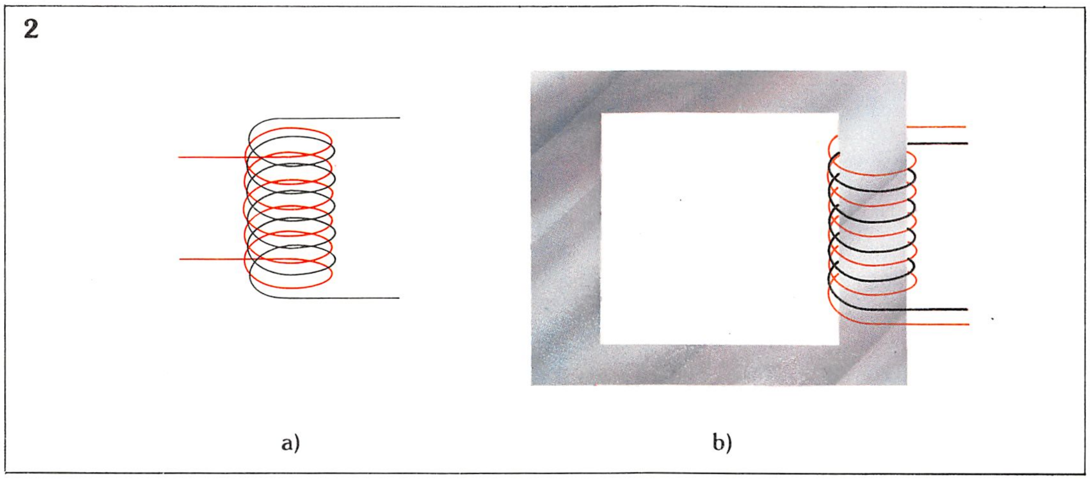
This shows that the induced voltage in the secondary coil is proportional to the rate of change of current in the primary coil.

In the same way, the voltage E_1 induced in the left-hand coil by a current I_2 through the right-hand coil is:

$$E_1 = -M\frac{I''_2 - I'_2}{T}$$

The important point to note in these expressions is that the mutual inductance is the same – whether the voltage is induced in the left-hand coil by a varying current in the right-hand coil, or vice versa. It is a matter of our own convenience which coil we label as

2



2. (a) A high degree of flux linkage can be produced in this air-cored inductor; **(b)** flux can also be concentrated by using a core of ferrite material, as in this small transformer-like construction.

primary and which as secondary.

A more practical definition of the unit of mutual inductance, incorporating induced voltage, can now be derived: a mutual inductance of 1 H exists between two circuits if an EMF of 1 V is generated in one circuit when the current in the circuit is changing at the rate of 1 As^{-1}. The practical value of this principle is enormous – we can generate voltages in a circuit without the need to make any direct electrical contact to it.

In the example in *figure 1*, we have assumed that *all* the flux generated by the current in the primary coil links with the whole of the secondary coil. However, if two inductors are constructed as in the examples, this is unlikely to occur because the coils are too far apart. Various constructions are used in an attempt to produce total flux linkage – *figure 2a* shows an example of an **air-cored inductor**. By using a core of ferrite material flux can be concentrated, producing a small **transformer** based on this construction, suitable for use in radio frequency circuits.

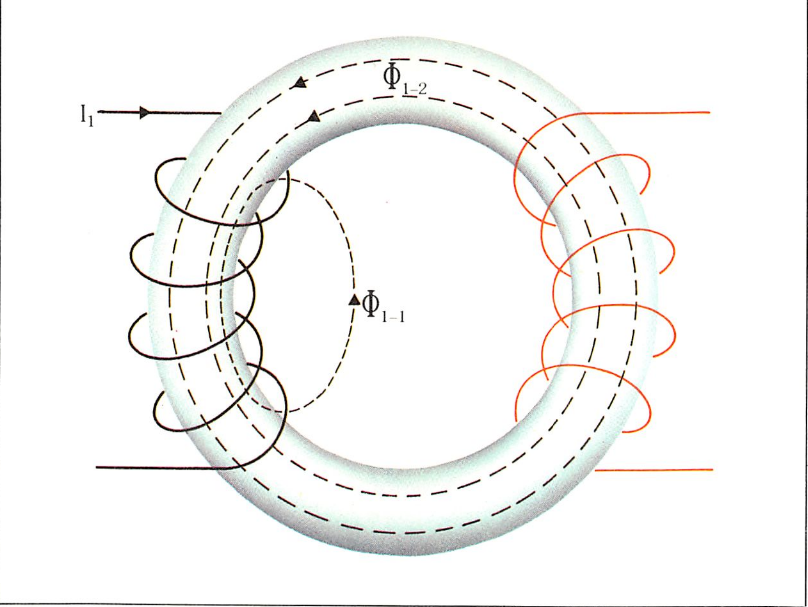
Cores in large transformers are usually made with a closed ring of ferromagnetic material, like those considered in the section on reluctance. An example is shown in *figure 2b*, and this type of construction might be found in a step-down transformer producing a low voltage output from the secondary with a higher, say, mains voltage input to the primary.

Leakage flux

Figure 3 shows an arrangement where most, but not all, of the flux ϕ_{1-2} generated by the current I_1 links with coil 1. The remaining **leakage flux** ϕ_{1-1}, links only with coil 1. A certain amount of self inductance, L_1, therefore exists with coil 1, as well as the mutual inductance M, between the two coils.

If the two coils are wound onto a larger core, so that they are a greater distance apart,

3



3. Two coils wound onto a larger core where most, but not all, of the flux generated by current I_1 links with coil 1.

the mutual inductance will decrease and the self inductance of each of the coils will correspondingly increase. Obviously, in a well designed mutual inductor, the coupling between coils should be as close as possible. The closeness of coupling is indicated by a **coupling coefficient**, k, which is a measure of the ratio between the mutual and self inductive effects, where:

$$k = \frac{M}{\sqrt{L_1 L_2}}$$

If the two coils are closely coupled so that all the flux from one coil links with the other, the coupling coefficient is 1. As the leakage flux increases due to the coils not being so closely coupled, the coupling coefficient falls until it becomes 0, when the two coils are a large distance apart. □

12 Sequential building blocks

Sequential circuits

Up to now we have looked at combinational building blocks, that is, circuits whose outputs depend upon the inputs. Each combination of input signals produces an output which is governed by the particular combinational blocks in the circuit and the way that they are connected: in other words, the combinational circuit **logic**.

However there is another important type of building block in digital electronics – the **sequential** circuit. Here, the output depends not only on the combination of inputs but also on the *sequence* in which they occur. Sequential circuits are capable of **storing** information, so an output depends on what has happened before, as much as it does on what happens next. This **memory** capability makes sequential circuits more complicated, and a different set of rules is needed to explain how they work.

Flip-flops

The basic unit of all sequential circuits is a **flip-flop**, of which there are many different types, all built from simple combinational gates.

The part of the circuit shown on the right in *figure 1* is an example of a flip-flop used as part of a larger circuit. The circuit as a whole is similar to the mechanical latch looked at, in general terms, in *Digital Electronics 1* (*figure 32*) and is known as a **transparent latch** – this is because signals are instantly passed from the data input D, directly to the true output Q, whenever the gating input G, is 1, as if the circuit was completely see-through. The **complemented** output, $\overline{Q}$, is the opposite of Q, and is produced as part of the circuit's internal operation.

However, when G is changed to 0, the output is latched in whatever state it was in when G changed, and it remains that way, irrespective of changes in D, until G returns to 1. Consequently, one bit of data can be stored in the latch while G is 0.

In some sequential circuits, a **clock** signal is used to synchronise data changes in various parts of the system (i.e. so that they all occur simultaneously) – these circuits are termed **synchronous**. **Asynchronous** circuits, on the other hand, do not use a clock signal.

How a gated latch works

The two AND gates shown on the left in

1. A transparent latch.

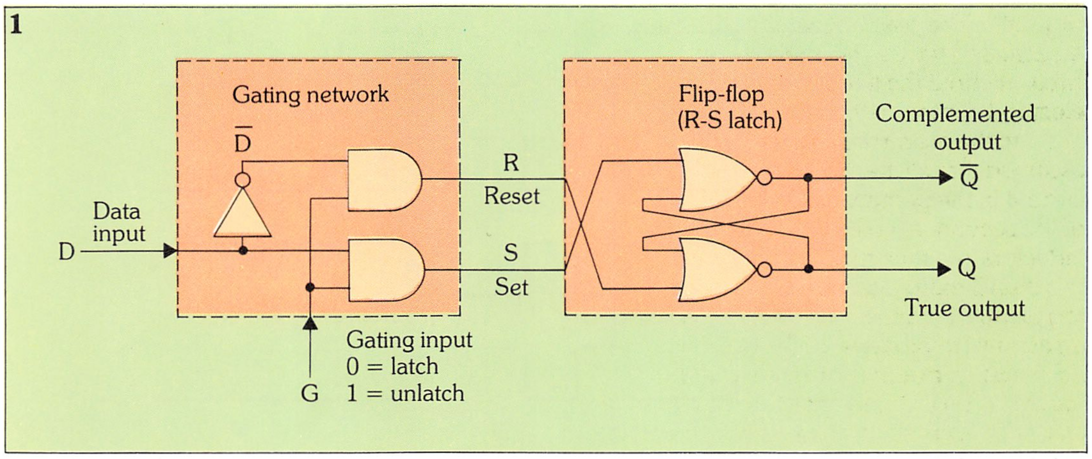


figure 1 form a **gating network**. The purpose of this circuit is to pass the signal D, through to lines R and S whenever G is 1. If the circuit is unlatched (when G is 1), R and S are complements of each other (because of the inverter); however when G changes to 0, the gating network makes both R and S go to 0 (the operation of the gating network can easily be worked out from the rules for the operation of AND gates).

The two NOR gates in the box on the right in *figure 1* form the most basic kind of flip-flop, an **R-S flip-flop**: this is the sequential part of the circuit, where the data bit is actually stored. In order to work out how an R-S flip-flop operates we use the rules for NOR gates: assume that the Q output is 1, the S input is 1, and the R input is 0. If the S input changes to 0, the Q output state of 1 remains. However, if the R input now changes from 0 to 1, the output Q will *flip* (i.e. change state) from 1 to 0. Only a change in the R input from 0 to 1 causes this to occur.

Similarly, if we consider the circuit when output Q is 0, only a change at input S of 0 to 1 causes the circuit to *flop* back, changing Q from 1 to 0.

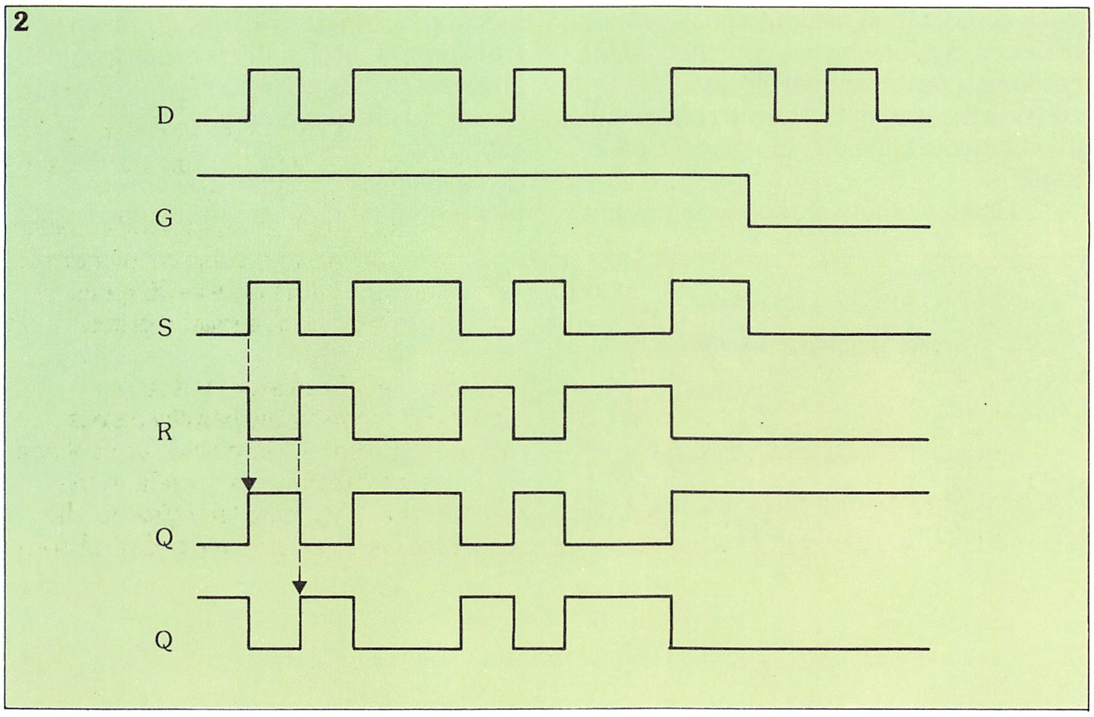
It is not advisable for R-S flip-flops to make both R and S go to 1 at the same time – this is because the outcome cannot be predicted.

Assuming that the gate in *figure 1* is open there is an inverter which complements the R and S inputs so that whenever R is 1, S is 0 and vice versa. This arrangement makes the S input change from 0 to 1 when the D input goes from 0 to 1, and the R input goes from 0 to 1 whenever D goes back to 0. As a result, the R-S flip-flop is continually flipping and flopping, with the Q output following the D input at every move. This is why the circuit is called transparent because it has no effect on the signal except to transfer it from input D to output Q, at least for as long as the G signal is 1. When G is 0, both the R and S inputs are 0 and the circuit is locked up or **latched**.

What makes this circuit sequential is the way the output of each NOR gate is fed back to the input of the other (this is called **cross-coupling**). As a result, the true output Q is the same as the input S, and in the circuit as a whole this makes Q the same as D, when G is 1; but changing D has no effect on the output when G goes to 0.

The working of the latch described here is shown in the **timing diagram** in *figure 2*, where the variations of the inputs and outputs with time are shown.

2

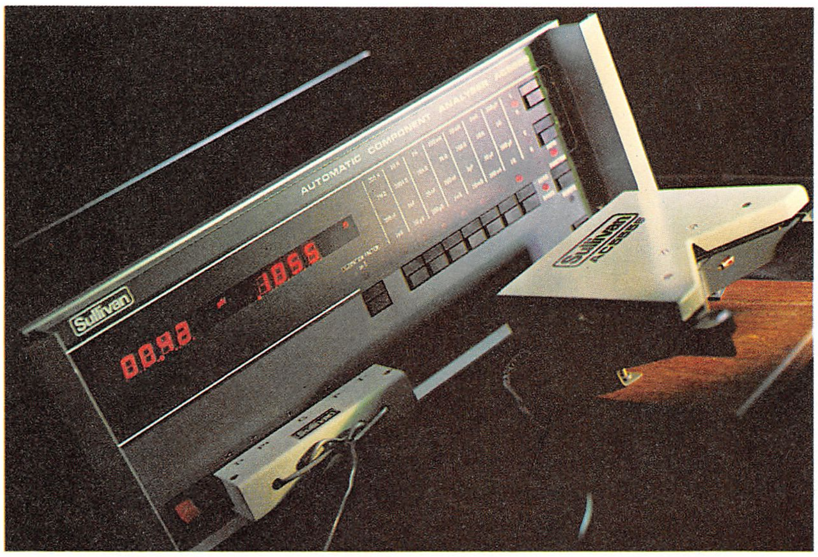


2. Timing diagram for the transparent latch shown in figure 1.

The name R-S for the NOR gate flip-flop is derived from the names normally given to its two inputs, **reset** and **set**. A momentary 1 in the R will *reset* the true output to 0, while a momentary 1 on the S line will *set* the true output to 1. Additional gating is provided on the inputs to the R-S flip-flop and the gate can be *latched up*, hence the term latch.

Circuits of this type are also termed **bistables** because each output can be in either of two states and will remain in that state (that is, is stable) until changed by a new input.

Below: many flip-flops are used in the circuitry of automatic testing equipment. Here, we see an automatic passive component analyser.



Paul Brierley

Flip-flops in a parallel register

All flip-flops are able to store and release one bit of information in response to various input signals, and it is this sequential property that makes them useful for constructing larger building blocks, for example, **parallel registers**. Any type of flip-flop could be used, but for the moment we'll examine a parallel register built from a number of simple latch circuits, such as the one in *figure 1*.

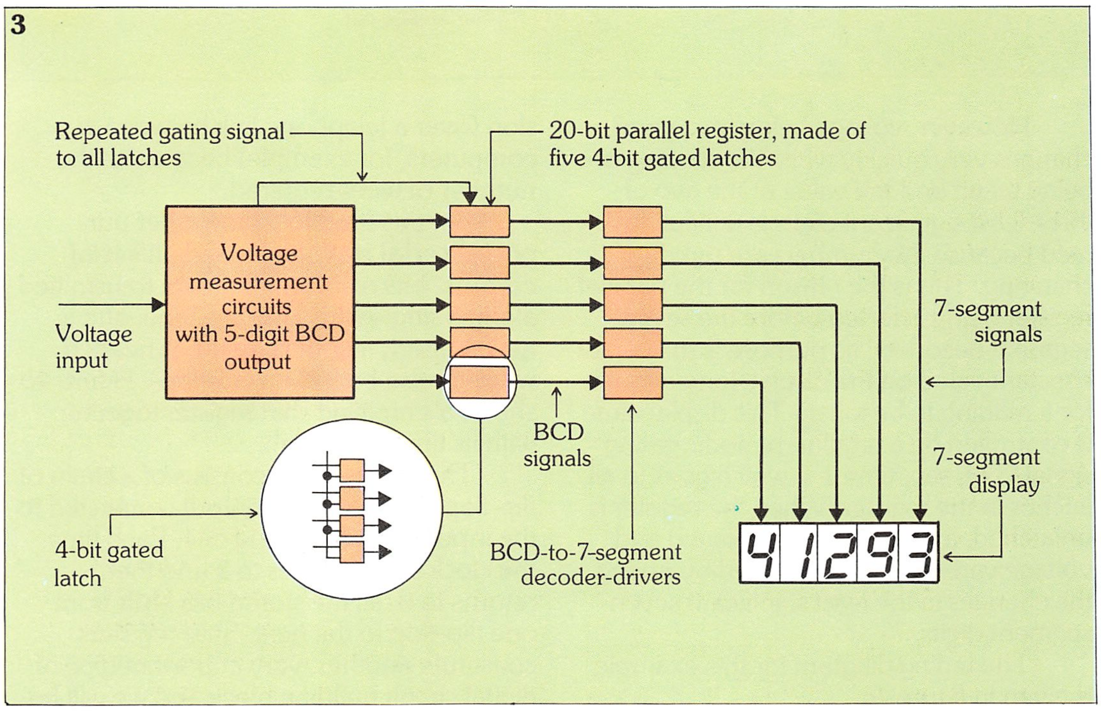
A parallel register comprises a group of flip-flops with common clock or gate inputs, but separate data inputs, thereby allowing a number of separate parallel bits of information to be either stored in, or released from, the register synchronously, i.e. at exactly the same time.

A typical application of parallel registers in a digital system is shown in *figure 3*, here, a digital voltmeter uses 20 gated latches grouped as a parallel register.

How a parallel register is used

Figure 3 illustrates how parallel data is transmitted from the large block of voltage measurement circuits on the left, to the display via a set of parallel registers. The large block is an **analogue-to-digital con-**

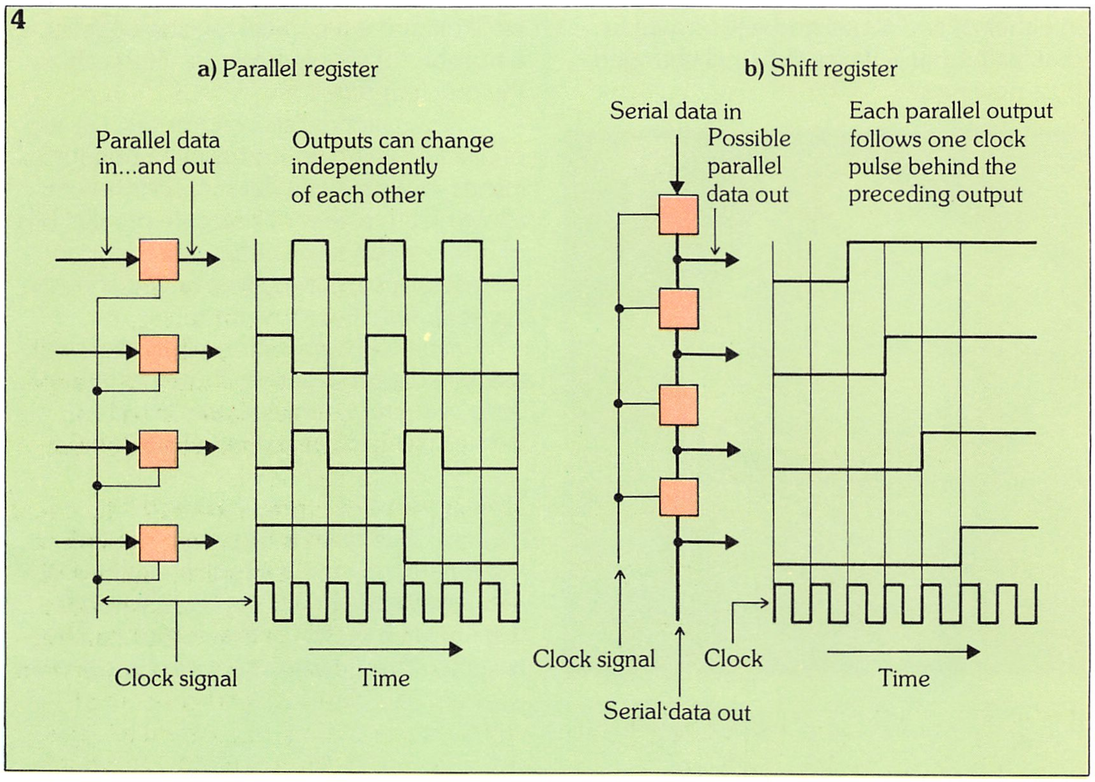
3


Repeated gating signal to all latches
20-bit parallel register, made of five 4-bit gated latches
Voltage measurement circuits with 5-digit BCD output
Voltage input
7-segment signals
BCD signals
7-segment display
4-bit gated latch
BCD-to-7-segment decoder-drivers
41293

3. Parallel data is transmitted from the analogue-to-digital converter on the left, to the display via a set of parallel registers.

verter, remember, this is a circuit block that converts the continuously varying voltage input (coming from the extreme left of the diagram) into parallel data, consisting of five digital words.

Each word is in binary coded decimal (BCD); each decimal digit is afterwards decoded by five BCD-to-seven-segment decoders which drive the display consisting of five seven-segment digits.



4
a) Parallel register
Parallel data in...and out
Outputs can change independently of each other
Clock signal
Time
b) Shift register
Serial data in
Possible parallel data out
Each parallel output follows one clock pulse behind the preceding output
Clock signal
Clock
Serial data out
Time

4. (a) Parallel register and its timing diagram; **(b)** shift register and its timing diagram.

Right: inspection of a microprocessor containing shift registers by microscope. Note the enlarged image on the VDU screen.

However, voltage being measured changes very quickly while the reading is being taken and the value of the two or three least significant digits is difficult to read because the numbers are rapidly changing. This is the reason for the parallel register being inserted before the seven-segment decoders: its purpose is to keep a constant value on the display long enough for a reading to be taken. This display time is controlled by a regular, periodic gating signal G of, say, time 1 s, which goes to all latches in the register. When the register is unlatched, a new value is displayed and voltage variations are observed by noting the changes in the least significant seven-segment digits.

The timing diagram for this example is given in *figure 4a*.

Shift registers

We have seen, in the example of the digital voltmeter, that each bit of parallel data requires a separate signal line, and that 20 individual latches are needed to transmit or store the information. This method is useful when the data is only travelling a short distance, however it would be very inefficient for long distance data transmission (over a telephone link between two computers, for example) because of the number of wires required.

For this (and for some other purposes) **serial transfer** is used. In **serial circuits**, bits of information are transmitted along a single wire, one at a time, and a new register, a **shift register**, is needed which is also built from flip-flops. *Figure 4b* shows a simplified shift register together with its timing diagram.

The shift register consists of a chain of flip-flops, each with its output connected to the input of the following one. Each time the clock signal passes to 1 and then returns to 0, all the stored bits **shift** from one flip-flop to the next. Shift registers constitute another, very important type of digital circuit building block and we will be discussing them in greater detail later on.

Using shift registers

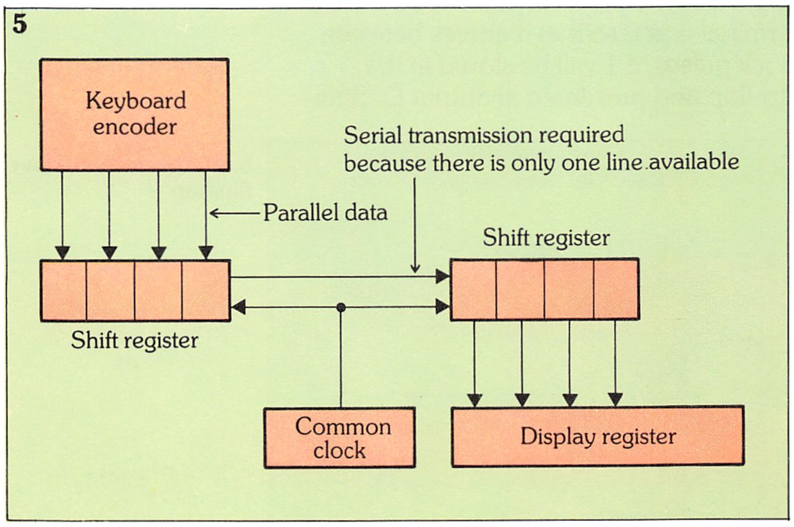
Although the shift register shown receives bits serially, the data can be *read out* in parallel; reading can only take place, however, when the register is not carrying out internal shift operations. Such a register is called a **serial in/parallel out** (SIPO) register. There are also **parallel in/serial out** (PISO) registers which reverse this procedure by accepting data in parallel form and transferring it out as serial data.

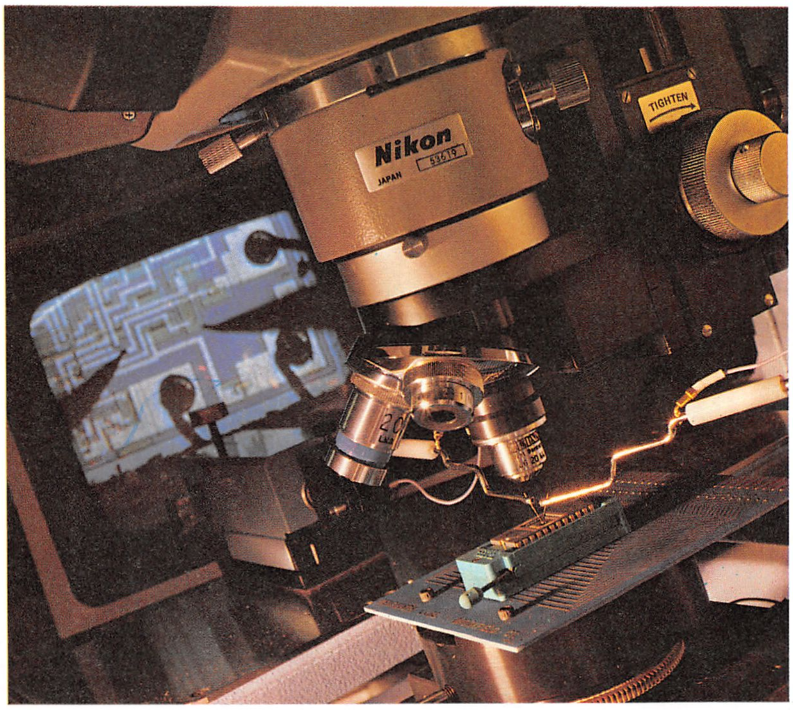
The example shown in *figure 5* illustrates how *data conversion* is performed by shift registers. Suppose that only one conductor (wire) is available for transmission of bits between a keyboard encoder and a display register. The two 4-bit shift registers illustrated allow 4-bit words to be transmitted from the encoder to the display register in series. First, the four bits of a BCD number are loaded *in parallel* into the left-hand shift register, then, the two shift registers are made to shift four places by a common clock signal. This causes the four stored bits to leave the left-hand register and enter the right-hand one. Here, the bits are read out in parallel and transferred to the display register.

Parallel data has thus been converted into serial form for transmission then back into parallel form for display.

5. A parallel in/serial out shift register. Parallel data has been converted into serial form for transmission, then back to parallel form for display.

5





How the system is synchronised

In *figure 5*, it is very important for the circuits on the left to be synchronised with those on the right: everything must happen at the right time or the data bits will be shifted in the wrong order, changing the value of the digital words.

Digital circuits must be carefully timed to ensure that all bits are in the right place at the right time. The simplest method of synchronising everything is to use a common clock signal (one that is simultaneously applied to all the building blocks), as can be seen from *figure 5*. This concept of **clocking** is extremely important in digital systems and occurs frequently. Although not all control signals to a flip-flop are clock signals, all control signals in a synchronised system are timed in relation to a master clock.

Another important factor must be considered. In shift registers, and in other circuits where data is transferred from one flip-flop to the next, the shift of the bits can become very complicated if all flip-flop outputs change at precisely the same moment. If this happens, the result is similar to both inputs to an R-S flip-flop being equal to 1 at the same time – the outcome is unpredictable, and this is not something you would want if, for example, you are trying to add two binary coded decimal numbers.

The solution to this problem is to use a different flip-flop, one which stores new data before the old data is changed.

Master-slave flip-flops

Figure 6 illustrates a two-stage flip-flop built on the **master-slave** principle, known as an **R-S master-slave flip-flop**. It comprises two identical units: one of which is the **master**, and the other is the **slave**. Each unit consists of a latch, similar to that in *figure 1*, but has two data inputs, and an inverter in the clock signal line to the slave. The inputs marked preset and clear have other uses which will be explained later.

The clock signal, $\overline{CK}$, for the slave unit is the inverse of the principal clock signal, CK, supplied to the master. This allows the process of *store before changing the output state* to take place. For example, the inputs S and R could be linked to the outputs Q and $\overline{Q}$ of another flip-flop controlled by the same clock signal. When the main clock signal changes from 0 to 1, the master stores the data presented to its S and R inputs; but when the clock returns to 0 once again, the $\overline{CK}$ signal goes to 1, so the slave stores new information received directly from the master's Y and $\overline{Y}$ outputs.

When these flip-flops are used in a shift register, all the flip-flops are ready to shift when the principal clock goes from 0 to 1, at what is called the **leading edge** of the clock pulse. Then the flip-flops complete the shift at the **trailing edge** of the clock pulse, that is, when the clock returns from logic 1 to logic 0.

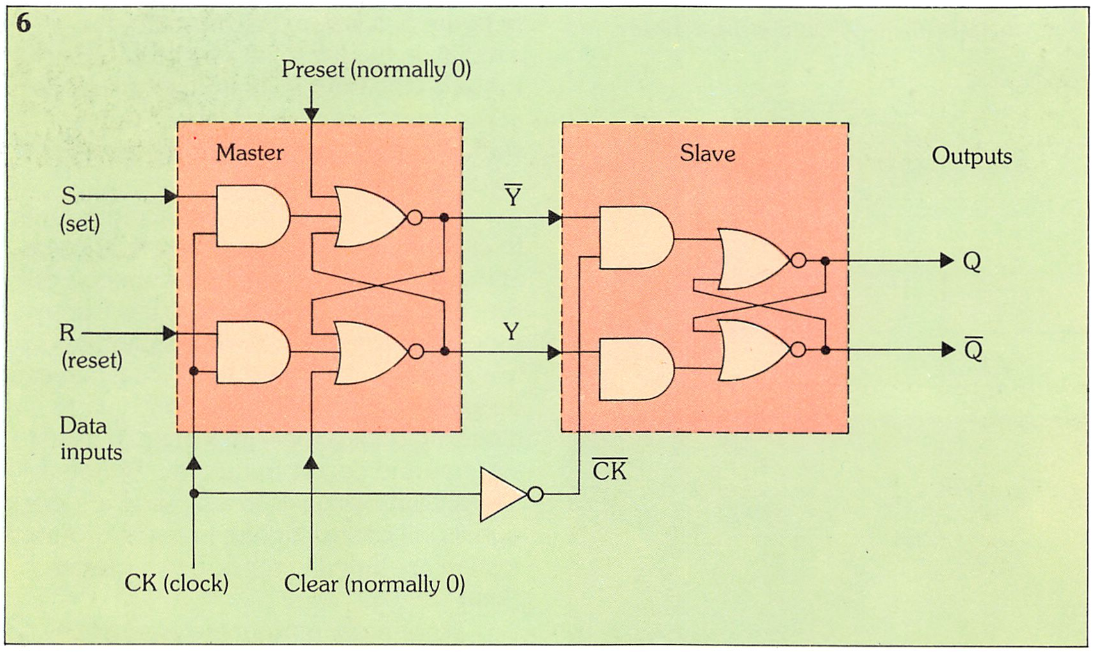
This master-slave principle allow bits to be shifted from one flip-flop to another by means of the same clock pulse, and any number of these flip-flops can be strung together.

Preset and clear inputs

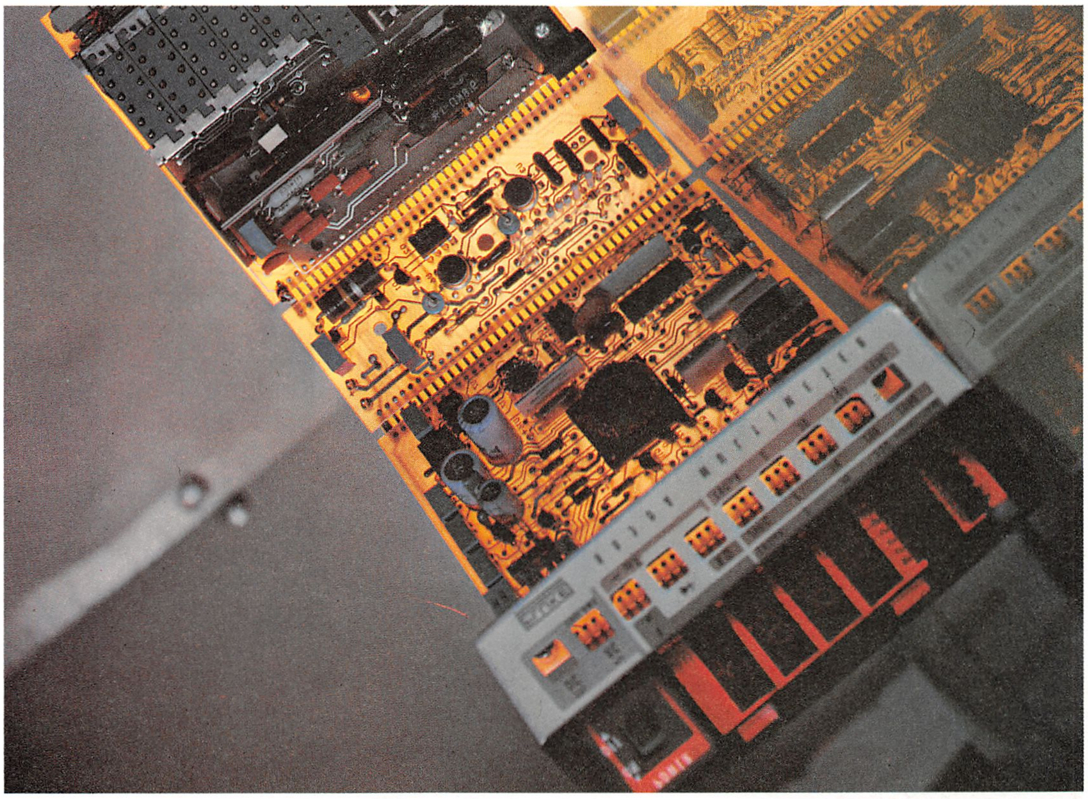
Earlier we mentioned two other inputs to the master-slave flip-flop: the **preset** and **clear** inputs. These are normally kept at 0, and therefore do not influence the operation of the circuit. However, if a momentary positive-going pulse (i.e. a pulse which goes from 0 to 1 then rapidly back to 0) is applied to the preset input while the clock terminal is at 0 so that it arrives between clock pulses, a 1 will be stored in the flip-flop and presented at output Q. Similarly, a momentary positive-going pulse applied to the clear input between two clock pulses will cause the flip-flop to store a 0.

Although not all flip-flops have these features, they can be useful. For example, some systems require old data to be *cleared* to 0, while others might call for all registers to be *preset* to 1. You may remember how, in *Digital Electronics 1*, the clear button of a calculator was used to wipe out all the data present in its registers. In effect, it was sending a clear signal to all the flip-flops in each register.

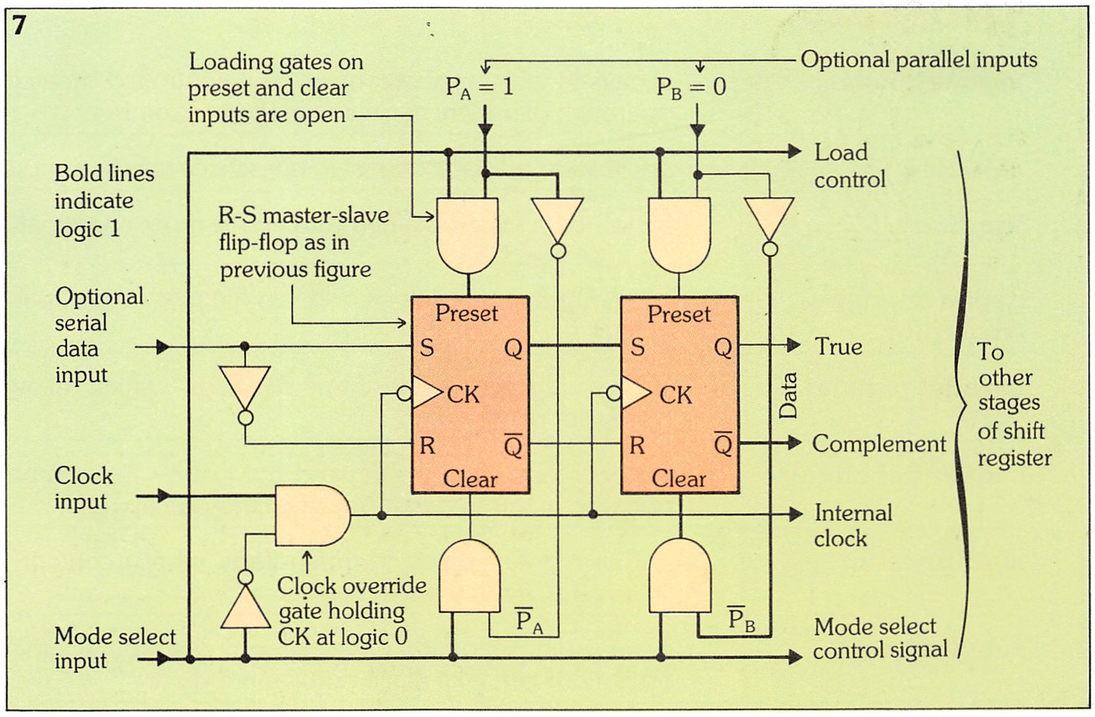
6



6. An R-S master-slave flip-flop



Right: inside a multimeter. Flip-flops would be used in this circuitry to store information.


7
Loading gates on preset and clear inputs are open
$P_A = 1$
$P_B = 0$
Optional parallel inputs
Bold lines indicate logic 1
Load control
R-S master-slave flip-flop as in previous figure
Optional serial data input
Preset
S
Q
CK
R
$\overline{Q}$
Clear
Data
True
Complement
To other stages of shift register
Clock input
Internal clock
Clock override gate holding CK at logic 0
$\overline{P}_A$
$\overline{P}_B$
Mode select input
Mode select control signal

7. Using preset and clear inputs for putting parallel data into a shift register.

How parallel data can be loaded using preset and clear

Preset and clear inputs can be useful for putting parallel data into a shift register, *figure 7* illustrates one method of doing this. Here, there are two shift register stages composed of R-S master-slave flip-flops like those shown in *figure 6*. Each flip-flop is indicated by the block symbol normally used for this type of circuit. The circle and triangle symbols placed at the clock inputs indicate that the flip-flop outputs change state when the clock passes from 1 to 0. This is called **negative edge triggering**, meaning that new data is loaded into the master while the clock pulse is 1, and is transferred to the slave as the clock goes from 1 to 0.

When the **mode select** input is at 0, the circuit operates simply as a normal shift register and accepts serial data; when the mode select input is at 1, parallel data can be entered via the preset and clear inputs because the effect of the mode select control input is to hold the internal clock signal at logic 0. (This is necessary for presetting and clearing these particular flip-flops.) At the same time, a logic 1 mode select signal unlatches the loading gates, allowing the flip-flops to be loaded in parallel. A logic 1 at a parallel input will operate the preset input of the corresponding flip-flop, setting the Q output to 1. A logic 0 will be inverted to 1 and will operate the clear input resetting the Q output to 0. As soon as the mode select input returns to 0, the shift register can once more be operated as a serial shift register.

This method illustrates one way of making a parallel in/serial out shift register, although there are many other ways of loading parallel data into a register. For example, some integrated circuit registers will only accept a logic 1 on each of the parallel inputs. To load a pattern of 1s and 0s into the register (say, the 4-bit word 1001), all the flip-flops must first be cleared to 0 by resetting all outputs. Then, if the correct pattern is presented to the parallel inputs, only the 1s are loaded into the first and fourth flip-flops, leaving the other two at 0. The result is that the register holds the 4-bit word, but it has taken two operations (master clear, then parallel load) to do it.

There are many other types of shift register, for example the **bidirectional shift register**, which can accept parallel data then shift either left or right.

Glossary

asynchronous	circuits in which operations are not controlled by a clock signal. One operation commences on completion of another
clocking	application of a clock signal to operate circuits in a set sequence
flip-flop	a basic data store obtained with two or more combinational logic gates
negative-edge triggering	when flip-flop operations occur as the clock signal passes from logic 1 to logic 0
parallel register	a data register, made from flip-flops, which allows synchronous storage of parallel data
sequential circuits	circuits whose outputs depend not just on the combination of inputs but also on the sequence in which they occur
shift register	a data register, made from flip-flops, whose contents can be serially shifted, i.e. moved one bit at a time
synchronous	circuits whose operations are controlled by a clock signal. Many operations can take place simultaneously
timing diagram	a diagram, showing inputs and outputs of a circuit in relation to control signals and time
transparent latch	a flip-flop circuit which can store or pass through data at its input depending on the applied control signal

14

SOLID STATE ELECTRONICS

FET amplifier circuits

Using FETs in amplifiers

In order to operate as amplifiers, FETs require biasing in a similar way to bipolar transistors. In *Solid State Electronics 12*, we found that the objective of bipolar transistor biasing is the establishment of an operating point within the operating limits of the transistor, and then the maintenance of that operating point despite variations in temperature and variations between individual devices; this is also true for FET circuits.

The technique of **equivalent circuits** will be used to aid the analysis of FET biasing circuits. This technique, freqently used in electronics, utilises simple circuit elements, under specified conditions, arranged with the same characteristics of a more complex circuit or device; the simple circuit is then used to predict the behaviour of the more complex system.

Fixed bias

Biasing a FET, for example an n-channel JFET, presents problems because the gate-source voltage, V_{GS}, must be negative. The bias voltage can be supplied by a battery, as in *figure 1a*: this is called a fixed bias. An equivalent circuit for this is shown in *figure 1b* and you can see that a negative gate-source voltage is maintained (because the source is more positive than the gate).

Current I_1 is the current due to the constant gate-source battery voltage. Current I_2 is the drain current of the FET given by the transfer characteristic. When the circuit is connected, these two currents must be equal, and they must also equal the current through the transistor, i.e. the drain current, I_D. So if current I_1 is drawn onto the transfer characteristic of the transistor, the I_1 and I_2 curves will intersect at the quiescent drain current.

Figure 1c shows extreme ranges of transfer characteristic between components of the same type, forming two curves I_{2a} and I_{2b}. In reality, the transfer characteristic of any FET of this type will lie somewhere inbetween these two extremes.

Current I_1 is shown as a vertical line originating from the battery voltage forming the fixed gate-source, V_{GS}. The line intersects the two extreme transfer characteristics at drain currents I_{D1} and I_{D2}, therefore the quiescent drain current of the circuit is *between* I_{D1} and I_{D2}.

Self-bias

By inserting a resistor, R_S, between the source of the JFET and ground, the drain current, I_D, creates a voltage across the resistor, holding the source at a higher voltage than the gate. The gate-source voltage, V_{GS}, is therefore negative. *Figure 2a* shows a typical arrangement.

If the drain current increases due to, say, a change in ambient temperature, the gate-source voltage becomes more negative, reducing the drain current. In this way the circuit tends to be self-regulating.

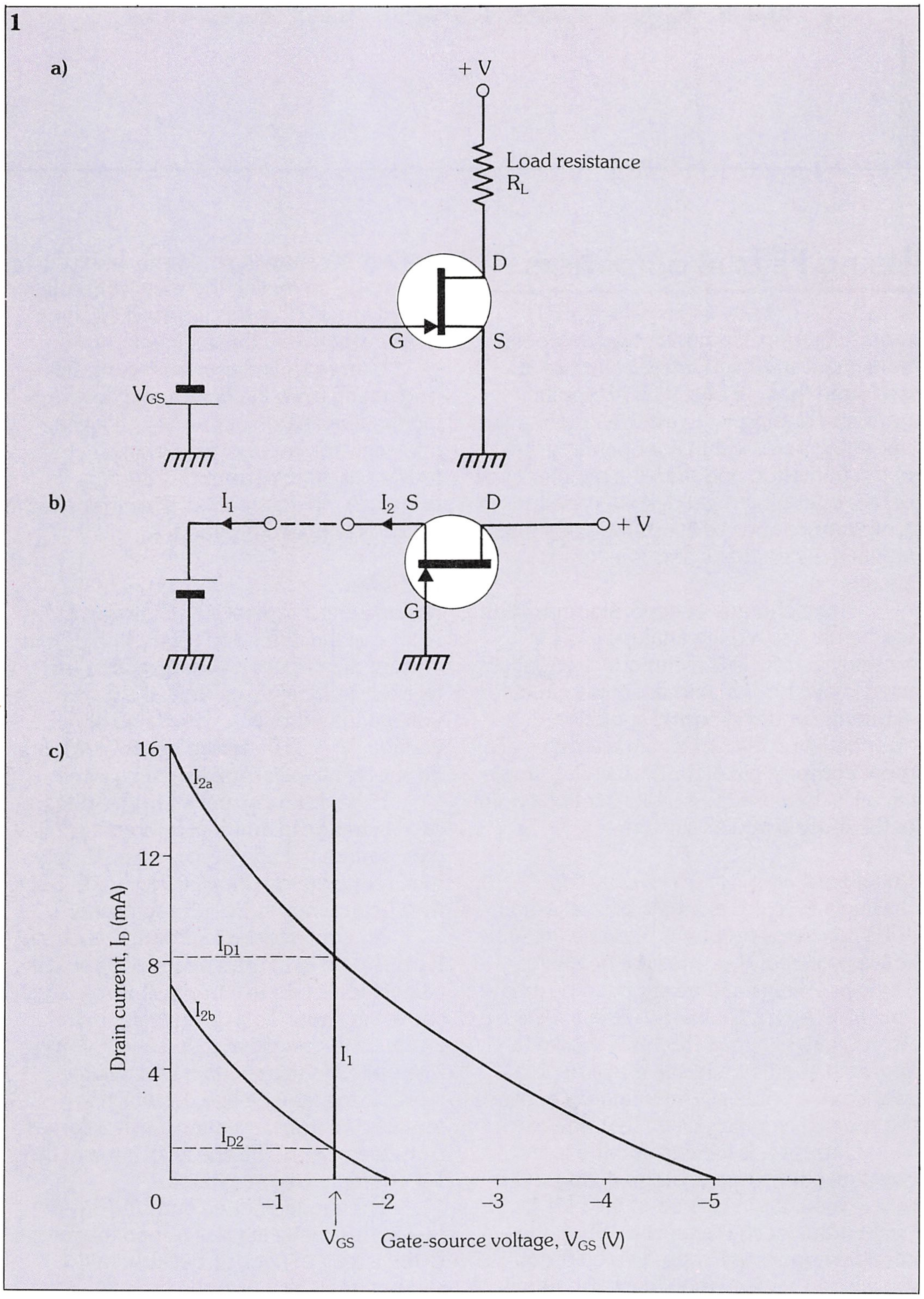
An equivalent circuit with the resistor R_S is shown in *figure 2b*, and a graphical calculation of drain current similar to that shown in *figure 1c* is given in *figure 2c*. Because the negative gate-source voltage is supplied in this circuit by the source resistor, the straight line, I_1, which intersects the transfer characteristics comes from the origin. The line is, of course, the I-V characteristic of a resistor.

The range of drain currents (I_{D1} to I_{D2}) in this circuit is smaller than the range in the circuit of *figure 1* because of the steeper angle at which the I_1 curve intersects the I_2 curves.

Bias combination

Obviously, limiting the range of operating drain currents helps to provide a more stable operating point. We can further limit the drain current range, thereby improving

1

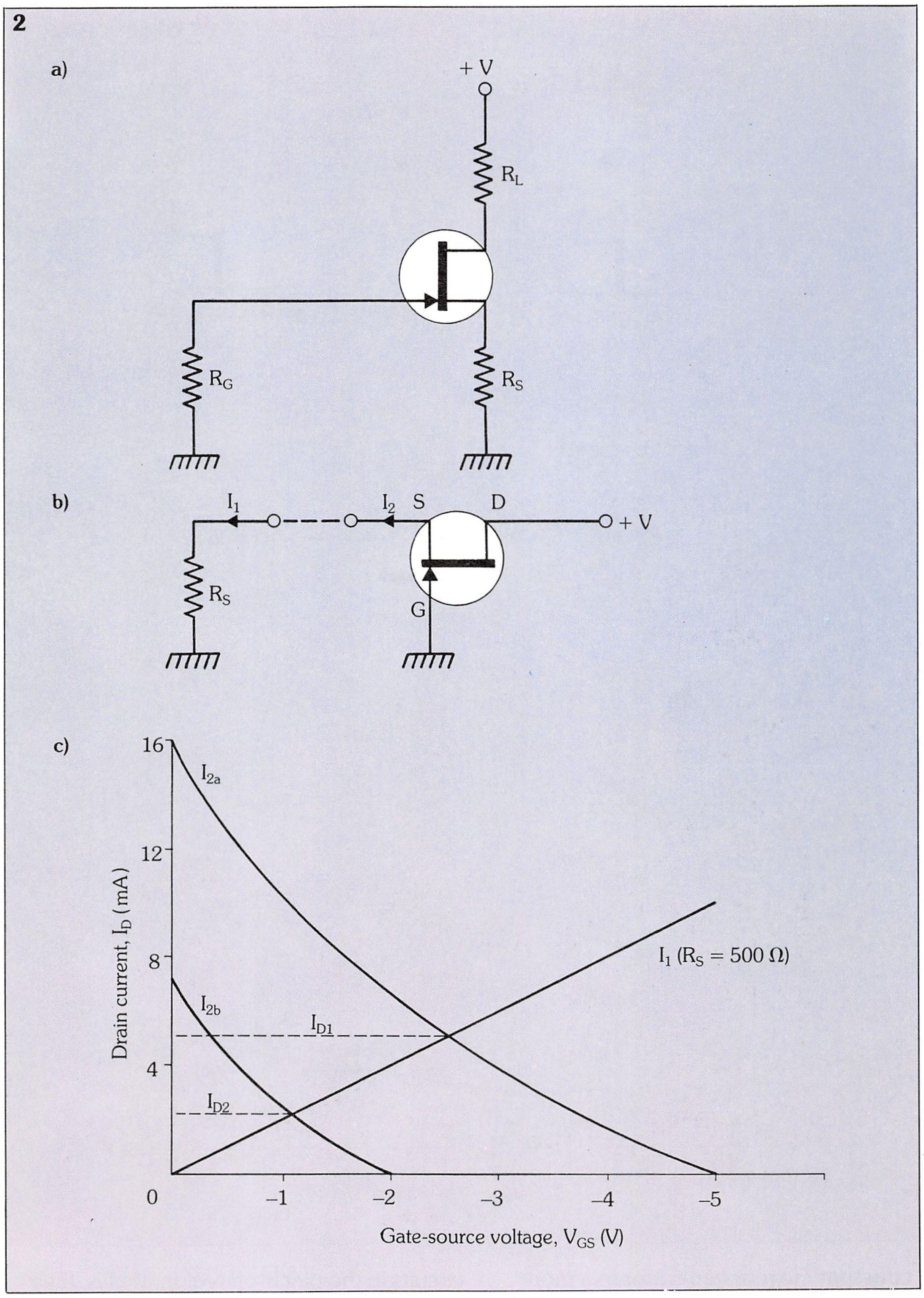


1. (a) Fixed bias– biasing an n-channel JFET with a battery; **(b)** an equivalent circuit; **(c)** I_D *vs* V_{GS} showing two extreme ranges of transfer characteristic between components of the same type.

circuit stability, by combining the two forms of biasing already discussed; two ways of doing so are shown in *figure 3a*. In the first circuit, a resistive potential divider holds the voltage at the FET gate at half the supply voltage; the second circuit has a **three-rail power supply**, providing voltages of +6 V, 0 V, and −6 V. This power supply has the same effect on the transistor as the resistive potential divider.

An equivalent circuit is shown in *figure 3b* while the transfer characteristic

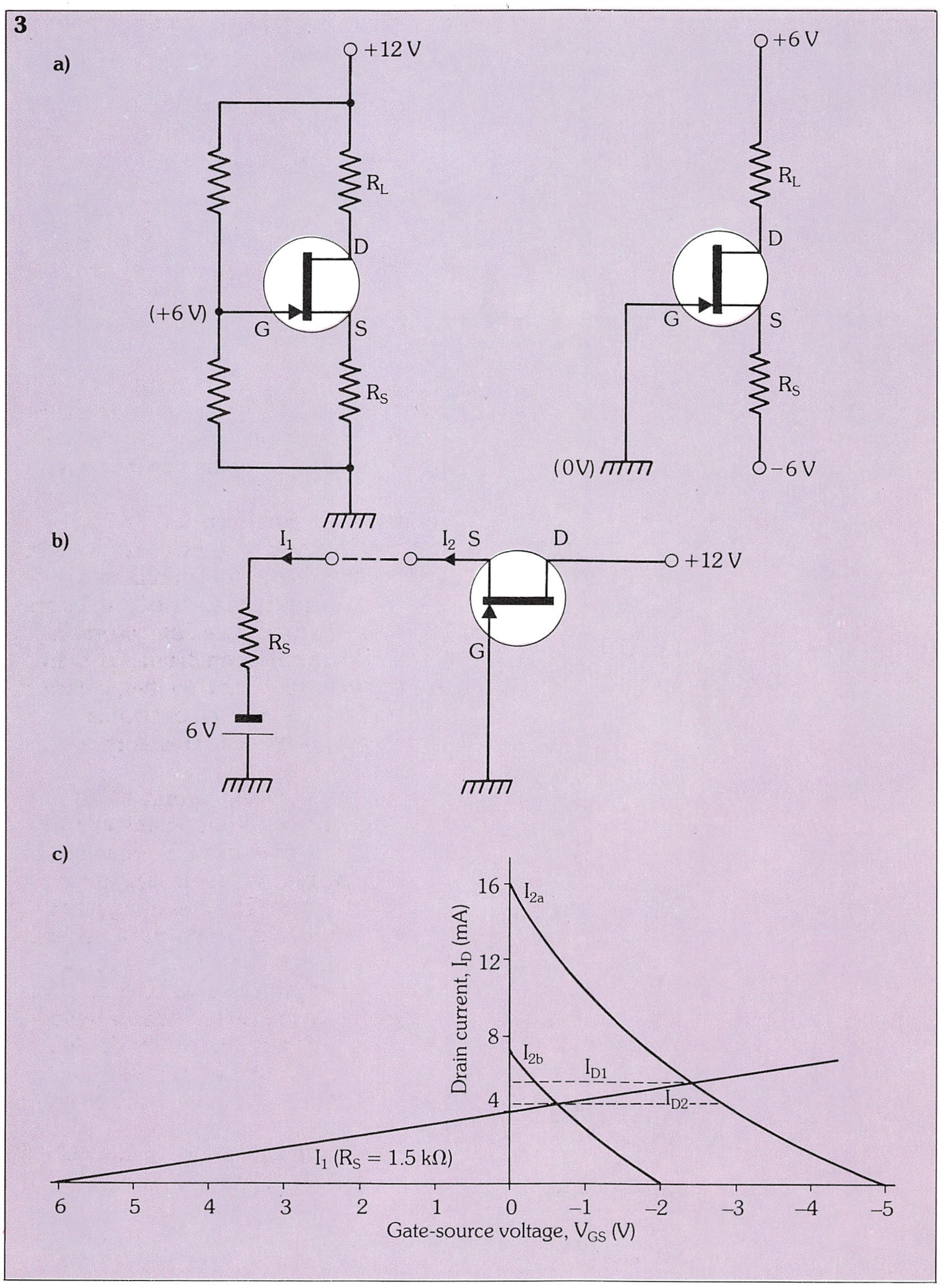
2. (a) Self-bias – a voltage is created across the resistor by placing resistor R_S between the source of the JFET and the ground; **(b)** an equivalent circuit; **(c)** I_D *vs* V_{GS} showing straight line, I_1 – this is the I-V characteristic of a resistor.



curves are given in *figure 3c*. Because the gate voltage is +6 V, the I_1 originates from the +6 V point on the voltage axis. The slope of the line corresponds to a source resistance, R_S, of 1.5 kΩ, so the line intersects curves I_{2a} and I_{2b} at drain current range levels I_{D1} to I_{D2}.

This range, however, is much lower than that of either of the bias circuits shown in *figures 1* and *2*.

An application of combined bias is shown in *figure 4*. Here the FET T_3 acts as



3. (a) Bias combination – in the circuit on the left, a resistive potential divider holds the voltage at the FET gate at half the supply voltage; in the circuit on the right a three-rail power supply replaces the potential divider; **(b)** an equivalent circuit; **(c)** transfer characteristic curves.

4. A differential amplifier where FET T_3 acts as a constant current generator.

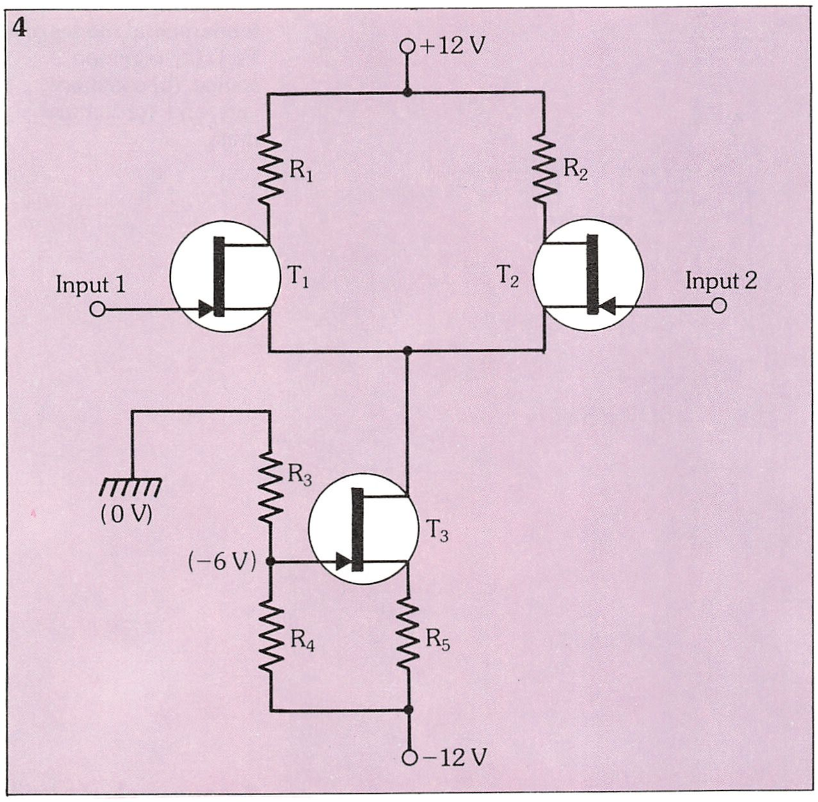
5. Drain-source resistance *vs* gate-source voltage for two p-channel FETs.

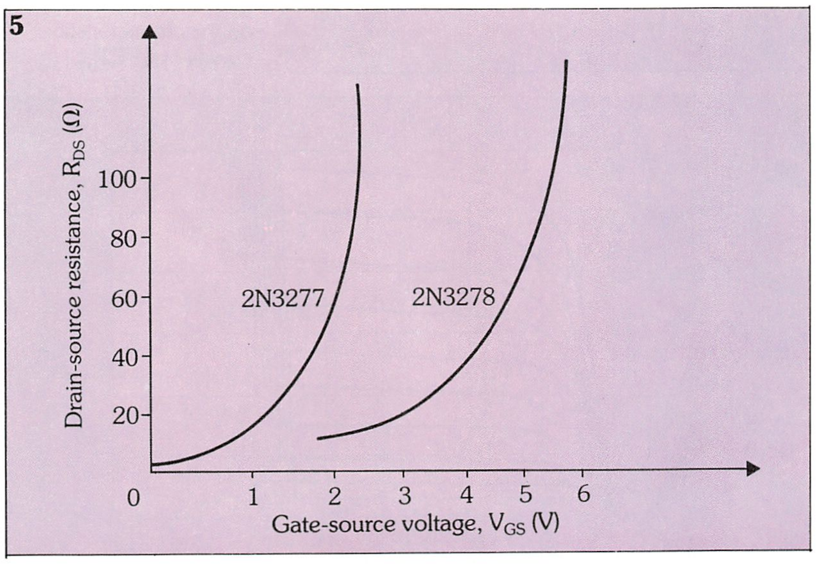
6. A typical application for a voltage controlled resistor is gain control.

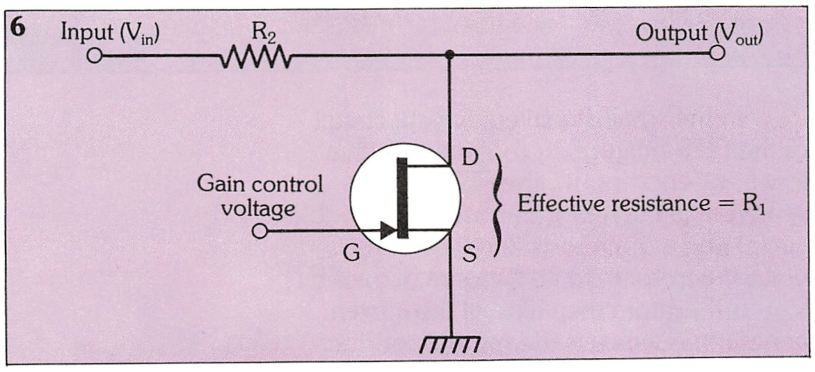
a **constant current generator** in a more complex circuit. The complete circuit is, in fact, a differential amplifier. The equivalent circuit and graphical analysis for transistor T_3 biasing are identical to those in *figure 3*.

FET voltage-controlled resistor

In general, FETs are used in AC amplifying circuits in the pinch-off region, that is, their operating points are situated somewhere along the upper, near horizontal regions of their output characteristics. If, however, the operating point of a FET lies in the straight part of the output characteristic, before pinch-off occurs, then the FET behaves like a resistor: its value varying according to

4


5


6


the applied gate-source voltage. Curves of drain-source resistance, R_{DS}, varying with gate-source voltage, V_{GS}, are shown in *figure 5* for two p-channel FETs.

A typical application for such a **voltage controlled resistor** is **gain control** in an amplifier. The simplest form of gain control is the variable potential divider shown in *figure 6*. Output voltage for a potential divider is given by:

$$V_{out} = \frac{R_1}{R_1 + R_2} V_{in}$$

and you can see that it depends on the value of R_1, the FET resistance. Thus, by varying the applied gate-source bias voltage, V_{GS}, the gain of the circuit varies.

Fundamental FET circuits

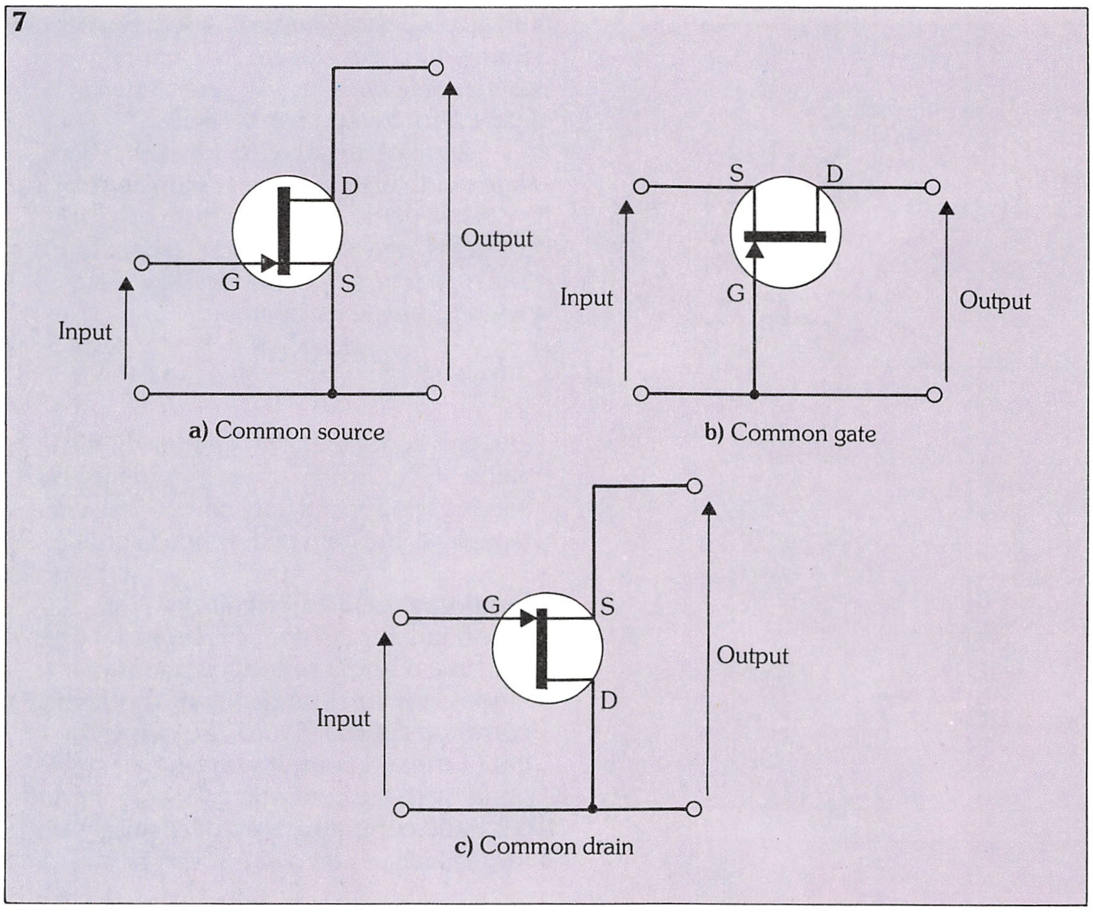
Like bipolar transistors, FETs can be operated in a number of configurations or modes; the three fundamental modes are **common source** (*figure 7a*), **common gate** (*7b*) and **common drain** (*7c*) corresponding to the common emitter, common base and common collector circuits, respectively, of bipolar transistors.

Defining the operating point

Load lines can be drawn on the output characteristic curves of a FET defining the operating point in the same way as for bipolar transistors. The operating point should be chosen in the pinch-off region. The characteristic curves can be considered (for our purposes) to be linear, and identical variations of gate-source voltage, V_{GS}, are equidistant. As long as the AC input signals are sufficiently small, the operating point will remain in the pinch-off region.

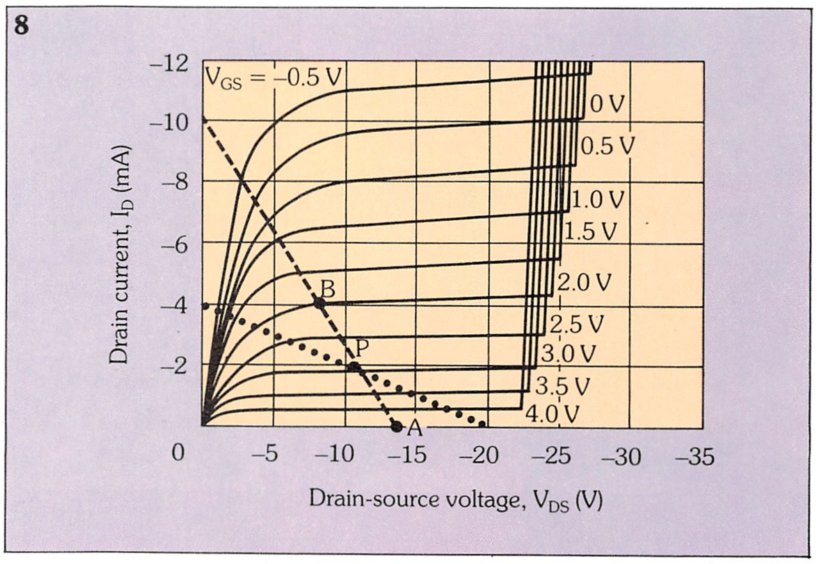
The load line depends on the overall load resistance. An example of a load line is shown in *figure 8* as a dotted line. Here the supply voltage is −20 V, and an example of a typical circuit which could use such a line is shown in *figure 9a*. In the circuit, the AC input voltage, V_{in}, is seen to be produced by a **voltage generator**, V_{vg}, and a series resistor, R_{vg}. This type of configuration is often used as an equivalent circuit of a **signal generator** (e.g. a stylus and cartridge of a record deck).

If we first consider the DC standing voltages and currents in the circuit (i.e., the



7. The three fundamental modes of a FET: (a) common source; **(b)** common gate; and **(c)** common drain.

8. An example of a load line (dotted line) drawn on the characteristic curves of a FET.



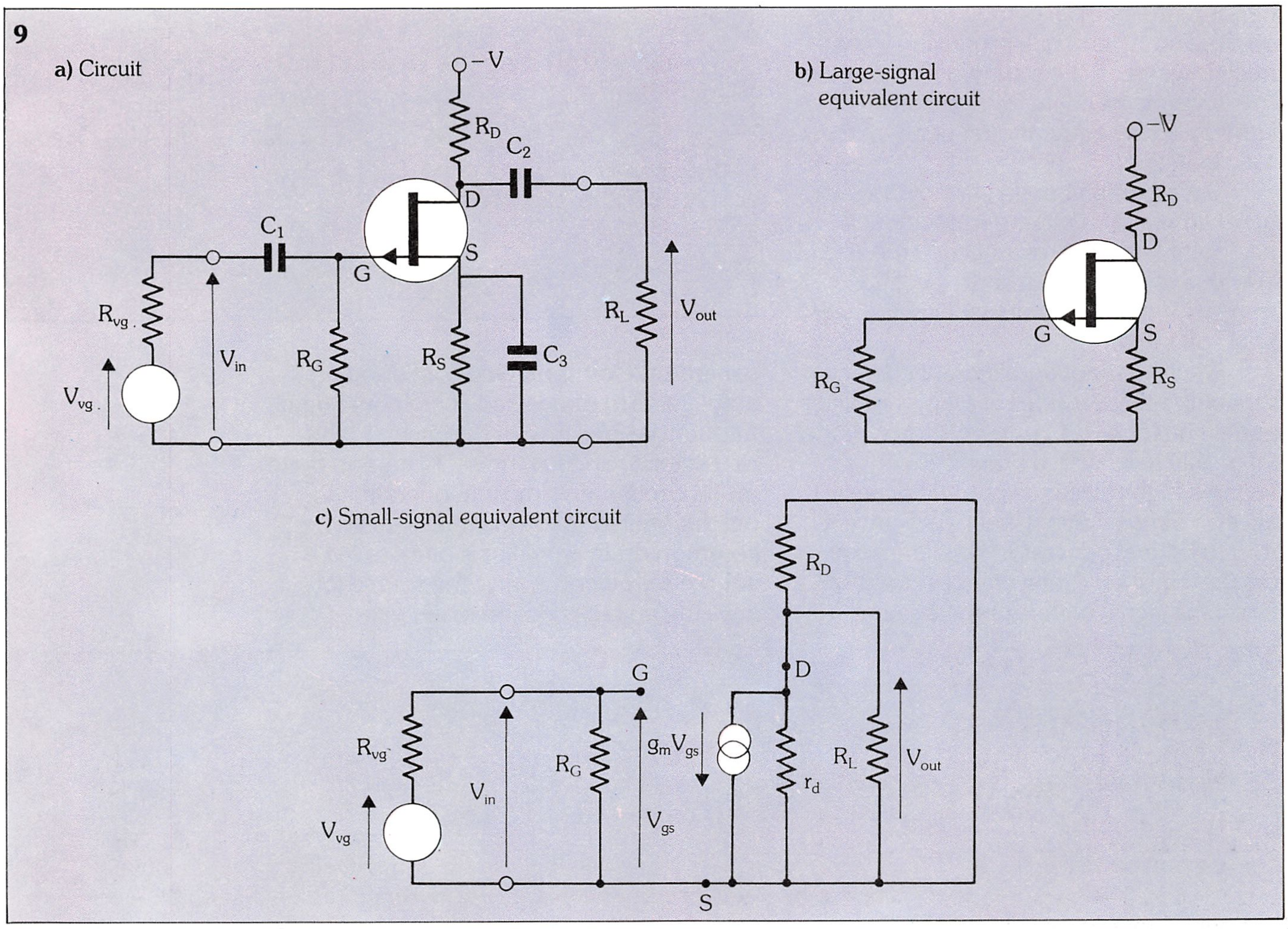
large-signals) then the capacitors C_1 and C_2 block all steady, continuous DC current from both preceding and following circuits, therefore all circuits before and after the transistor can be ignored. Capacitor C_3 also has no effect on the large-signals as it represents an open circuit at DC.

We are left with the equivalent circuit shown in *figure 9b* where the overall, static, load resistance is the sum of resistors R_D and R_S (in the example this is 5 kΩ) which can be used to draw a **static** load line (the dotted line in *figure 8*). On this we choose an operating point, point P, about in the middle.

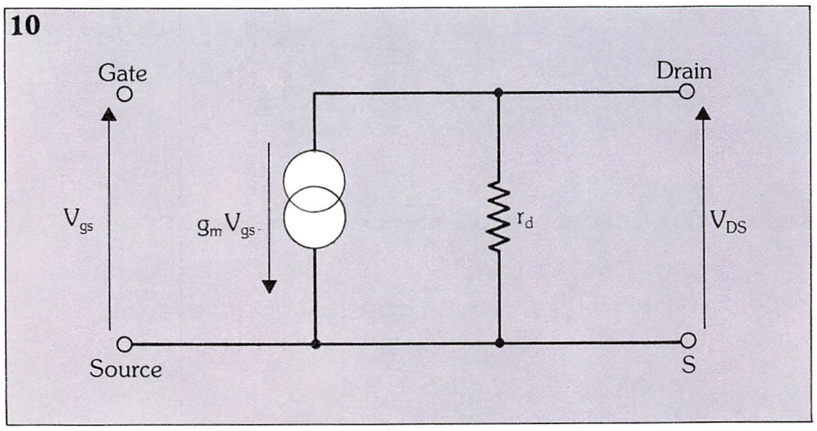
A FET, indeed any semiconductor component, is essentially a non-linear device, i.e., its output characteristic is *not a straight line.* In order to analyse FETs for small variations of current and voltage, i.e. small-signals, using Ohm's law and other electrical circuit theories, the equivalent circuit has to be linear; obviously, this linear equivalent circuit is only an approximation of the real circuit but adequate for small variations.

In the small-signal equivalent circuit of the FET in *figure 10* (operating with a low frequency input), the FET is represented by a **current generator** (the double circle) and a drain resistance, r_d. The voltage across drain and source of the FET, V_{DS}, is therefore the value of the current generator – which is the mutual conduct-

9



10



9. (a) Typical circuit which could use a load line; (b) large-signal equivalent circuit; **(c)** small-signal equivalent circuit.

10. Small-signal equivalent circuit.

ance, g_m, of the FET multiplied by V_{gs}.

The resistances between gate and source, and gate and drain, are considered to be infinite (the gate-channel p-n junction is reverse biased, remember), consequently no connection is shown between gate and source, or gate and drain.

Other components, particularly capacitors, could have been added, however, as this FET is operating at low frequency, they are not necessary. (If the FET was operating at high frequency, capacitors would be added.)

If we now consider the AC voltages and currents (i.e., the small-signals) a different **dynamic** equivalent circuit of the complete amplifier (*figure 9c*) can be drawn. The effect of the signal generator cannot be ignored now, nor can the effect of the following circuits, R_L. Capacitor C_3 acts as a short-circuit to AC signals so produces a short-circuit between the FET source terminal and ground. We have replaced the symbol for the FET by its small signal equivalent circuit of *figure 10*

Part of a power supply's effect on an equivalent circuit is to act as a short-circuit. This is because the internal resistance of a power supply is very low, and is therefore approximated in an equivalent circuit diagram to 0 Ω. The effective dynamic load resistance of the circuit in *figure 9a* is therefore the *parallel* combination of resistors R_D and R_L and r_d – producing a value much lower than the static load resistance. The dynamic load line (the broken line in *figure 8*) is consequently steeper than the static load line.

The operating drain current range of this example circuit varies from about 4 mA (point B) to approximately 0 mA (point A) with a range of drain source voltage from −8 V to −12 V, all in the near-linear pinch-off region.

A summary of the different characteristics of the three modes of a FET amplifier is given by *table 1*. Common source mode is typically used with a signal generator having a high resistance output because it takes an almost infinitesimal load current from the signal generator. Bipolar transistors used in such circumstances could distort the signal by **loading** the signal generator. Common gate configuration amplifiers are often used at high operating frequencies because of the circuit's inherent stability and low noise. Common drain mode circuits are similar in operation to bipolar emitter follower circuits. In fact, a common drain amplifier is often called a **source follower** because the source follows the voltage applied to the gate.

Table 1
Characteristics of the three modes of a FET amplifier

Configuration	Voltage gain	Input resistance	Output resistance
common source	greater than 1, inverted	extremely high	high
common gate	greater than 1, non-inverted	low	very high
common drain	less than 1, non-inverted	extremely high	low

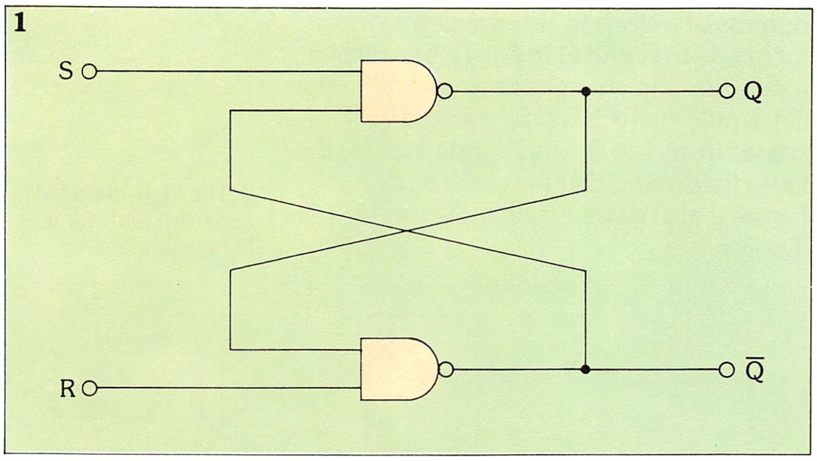
Glossary

combined bias	method of defining a FET circuit's operating point by biasing the FET using a combination of fixed and self bias
common drain	FET circuit configuration in which the transistor drain is effectively common to both input and output. Also known as a source follower, the configuration is similar to the bipolar transistor common collector circuit (emitter follower)
common gate	FET circuit configuration in which the transistor gate is effectively common to both input and output. Similar to the common base bipolar transistor configuration
common source	FET circuit configuration, similar to the common emitter bipolar transistor mode, in which the source is effectively common to both input and output
dynamic load line	the load line obtained on a FET's output characteristic curves when the circuit is analysed using AC (small-signal) input voltages
equivalent circuits	method of simplifying electronic circuits, by replacing non-linear devices, e.g. semiconductors, with linear elements, e.g. resistors, so that simple circuit theory can be applied
fixed bias	application of a fixed gate-source voltage to a FET biasing the transistor to operate as an AC amplifier
self bias	biasing a FET by including a source resistor. The voltage developed across this resistor ensures the gate-source p-n junction is reverse biased

13

DIGITAL ELECTRONICS

Flip-flops



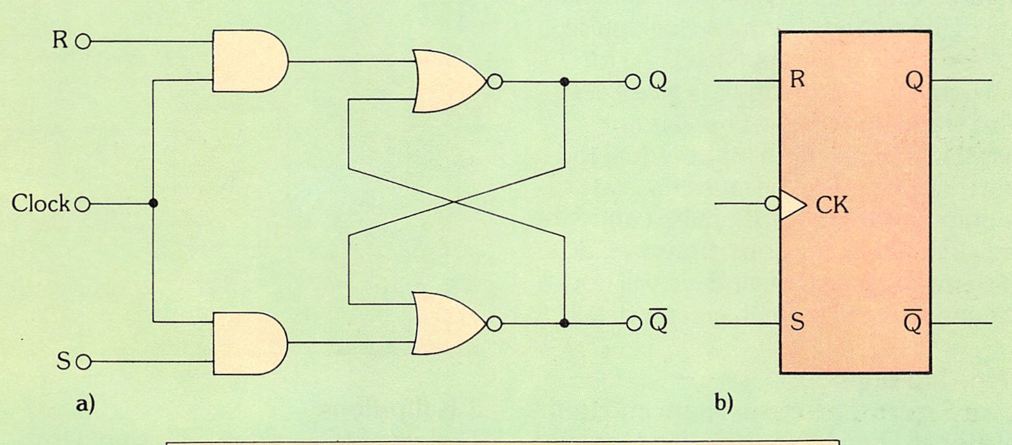
1. **An R-S flip-flop** built from two NAND gates.

Different types of flip-flop

In *Digital Electronics 12* we looked at the R-S flip-flop. This is the most basic type of flip-flop, being the building block for larger, more complicated circuits.

We found that both the R and S inputs of a NOR gate R-S flip-flop should not be set to 1 at the same time because the output would be impossible to predict.

R-S flip-flops can also be formed from NAND gates. Here, the outputs are made to change by taking one input to 0 while the other is held at 1. However, the



2. **(a) A clocked R-S flip-flop; (b)** its circuit symbol; and **(c)** its truth table.

c)

Truth table for a clocked R-S flip-flop				
Inputs		Outputs		Comments
R	S	Q_n	Q_{n+1}	
0	0	0	0	} No change
0	0	1	1	
0	1	0	1	If S = 1 when R = 0, Q → 1
0	1	1	1	No change
1	0	0	0	No change
1	0	1	0	If R = 1 when S = 0, Q → 0
1	1	0	?	} Output is uncertain
1	1	1	?	If R = 1 when S = 1

output is again unpredictable if both inputs are taken to 0 at the same time. *Figure 1* illustrates an R-S flip-flop built from two NAND gates; this can be proven to work by drawing a NAND gate truth table and working through the circuit.

Clocked R-S flip-flops

A variant of the R-S flip-flop is the **clocked R-S flip-flop** shown in *figure 2a*; its circuit symbol is shown in *figure 2b*. Although this circuit appears similar to the latch circuit seen earlier, it lacks the inverter element that ensured input complementarity. The clocked R-S type obeys the same rules as a NOR gate R-S flip-flop, but the outputs only change when the correct combination of inputs is present *and* when the clock pulse goes to 1. The gate circuit is the same as the NOR gate latch circuit, however the gates are opened and shut by a regular clock pulse instead of an irregular latching signal.

The truth table for this circuit is shown in *figure 2c*. As this is a clocked circuit, the state of the outputs before a clock pulse (Q_n) and after a clock pulse (Q_{n+1}) for each combination of inputs is given. As with a standard R-S flip-flop, it is not advisable to make both inputs equal to 1 before a clock pulse, because the state of the outputs after the clock pulse cannot be predicted. Other flip-flops, however, do not share this problem, and we will now examine some of them in greater detail.

D-type filp-flops

Figure 3 gives the logic diagram and truth table for all D-type flip-flops (the D stands for single D input). The true output, Q, follows the state of the D input, but note that the outputs change only at the moment in which the clock passes from 1 to 0 (negative-edge triggering). Changes at D have no effect when the clock is not changing, so the data presented at the D input has to be synchronised with the clock.

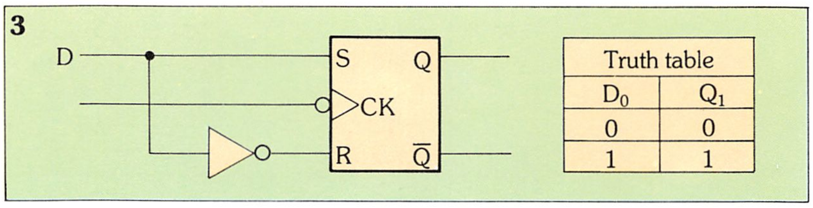
There are many ways of building a D-type flip-flop, one of which is from an R-S flip-flop and an inverter; preset and clear inputs could also be provided.

T-type flip-flop

A T-type flip-flop (T stands for toggle) has no data input at all; its outputs change state with every clock pulse, i.e. either when the clock signal goes to 1 or when it returns to 0, depending on the design of the particular flip-flop. In other words the flip-flop toggles at every clock pulse, just as a light switch toggles on and off each time it is operated. *Figure 4* illustrates how to make a T-type flip-flop by connecting the two outputs of a clocked R-S flip-flop to its inputs (Q to R and $\overline{Q}$ to S). This particular toggle flip-flop changes state each time the clock pulse (which is now marked by T) passes from 1 to 0 – this is indicated by the little circle marked at the clock input. *Figure 4* also gives a truth table for this flip-flop.

3. The logic diagram and truth table for a D-type flip-flop.

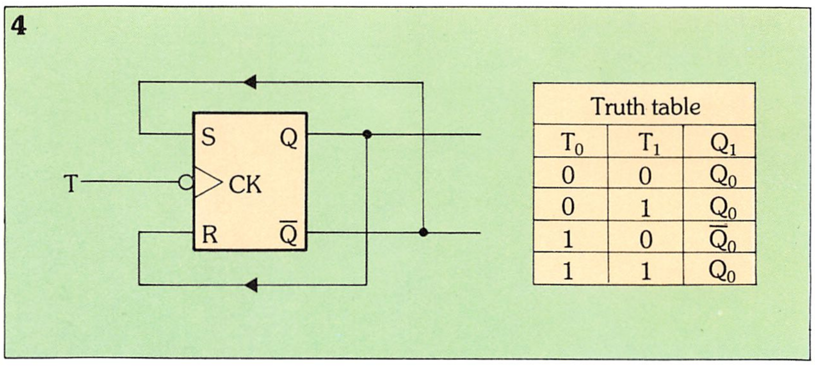
3



Truth table

D_0	Q_1
0	0
1	1

4



Truth table

T_0	T_1	Q_1
0	0	Q_0
0	1	Q_0
1	0	$\overline{Q}_0$
1	1	Q_0

4. The logic diagram and truth table for a T-type flip-flop.

J-K flip-flops

The advantage of the J-K bistable (or *flip-flop*) shown in *figure 5* is that, like the R-S master-slave flip-flop discussed in *Digital Electronics 12*, it also avoids the indeterminate output condition which accidentally occurs with basic R-S flip-flops when both of its data inputs are taken to one at the same time. As you can see, it is similar to the T-type flip-flop except that the feedback from outputs to inputs goes via a pair of AND gates. This means that the input to the R-S part can never be R=1 and S=1. The true input is called J and the complementary input K.

The truth table in *figure 5* applies to all J-K flip-flops and indicates operation exactly like clocked R-S flip-flops, except when both the J and K inputs are at 1. When this happens, the outputs simply toggle, like a T flip-flop. In this way the uncertain condition (indicated by ? in our truth tables) is avoided.

J-K flip-flops are both versatile and widely used. Sometimes, a particular application requires the J or K inputs to be AND functions of various signals, these can be introduced through extra inputs to the AND gates at the front part of the flip-flop, in which case they are numbered J1, J2, K1, K2 and so on.

There are numerous variations of the four basic types of flip-flop (R-S, D, T, and J-K) but most of the differences are related to the inputs and the method of clocking, by which we mean what happens, exactly, when the clock passes from 0 to 1, from 1 to 0 or when it is at 0.

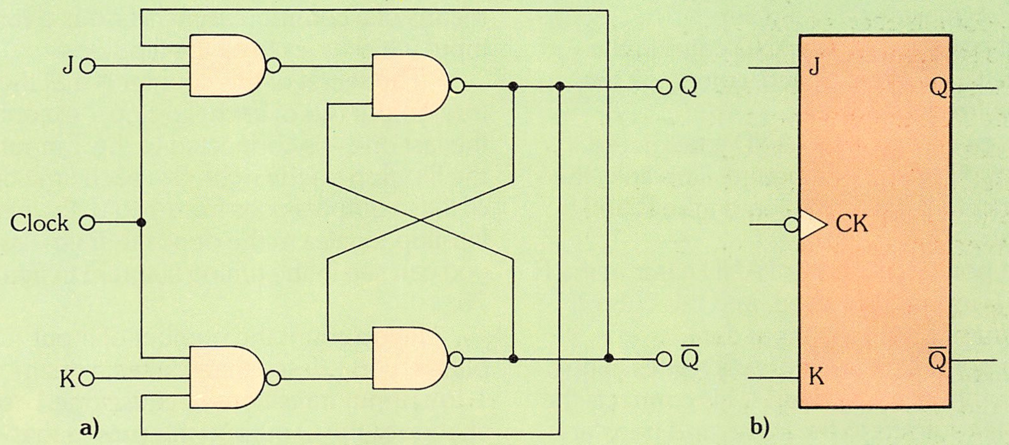
We shall now look at the process of clocking in flip-flops so that we may then be in a position to understand any new types we meet.

Edge or level-triggered flip-flops

Most flip-flops are synchronous devices, that is, instead of supplying an output signal immediately after a change at the inputs, they wait until a clock pulse arrives. Only then do they respond and give a new logic state in output. The term **triggered** means the same as clocked when used to describe the operation of a flip-flop.

There are three possible ways of triggering a flip-flop. A **level-triggered** flip-flop is sensitive to the logic state of the clock, rather than changes in its state; that is to say, triggering occurs when the clock is 1 or 0, but not when its state is changing. The logic state of the clock input (be it 0 or 1, according to the type of the flip-flop) transfers information from the input to the output. A **positive-edge triggered** flip-flop reacts to the leading edge of the clock signal. The transfer of information takes

5



Truth table for a J-K flip-flop

Inputs		Outputs		Comments
J	K	Q_n	Q_{n+1}	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	No change
0	1	1	0	If J = 0 when K = 1, Q → 0
1	0	0	1	If J = 1 when K = 0, Q → 1
1	0	1	1	No change
1	1	0	1	If J = 1 when K = 0,
1	1	1	0	Q always toggles

c)

5. (a) The J-K bistable (or flip-flop); **(b)** its circuit symbol; and **(c)** its truth table.

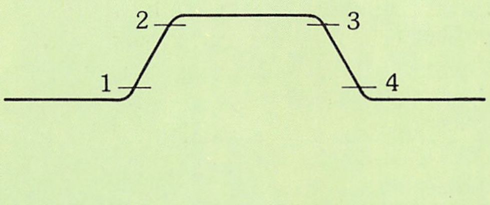
place on the positive edge (also called the rising edge) of the clock pulse.

The transfer of information in a **negative-edge triggered** flip-flop takes place on the negative or falling edge of the clock pulse. It is important that the data inputs are only changed when the clock pulse is not active. In any logic block, the input data or information must not be changed or altered during the period of a clock pulse. In logic blocks sensitive to the *edge* of the clock pulse, the input data must be stable for a certain period (the set-up time) before the leading or trailing edge.

A master-slave flip-flop, as we have seen, is a circuit which has two flip-flops, a master and a slave. The master receives the information on the positive edge of a clock pulse and transfers the information to the slave (the output flip-flop) on the negative edge of the pulse.

Figure 6 shows that this sequence of data transfers – from input to output – takes place in four stages. If you refer back to *figure 6* in *Digital Electronics 12*, you will be able to trace the sequence as follows for an R-S master-slave flip flop:
1) at point 1, the two AND gates in the slave flip-flop are closed, cutting off the slave from the master;
2) at point 2, the two AND gates in the master flip-flop are opened. This links the master to the information on the R and S inputs;
3) at point 3, the pair of AND gates at the R and S inputs are closed, and this cuts off the master from the input data;
4) at point 4, the pair of AND gates at the slave inputs are opened. This connects the master outputs to the slave, and transfers the information to the slave flip-flop outputs.

6



6. Sequence of data transfers – from input to output – in a master-slave flip-flop.

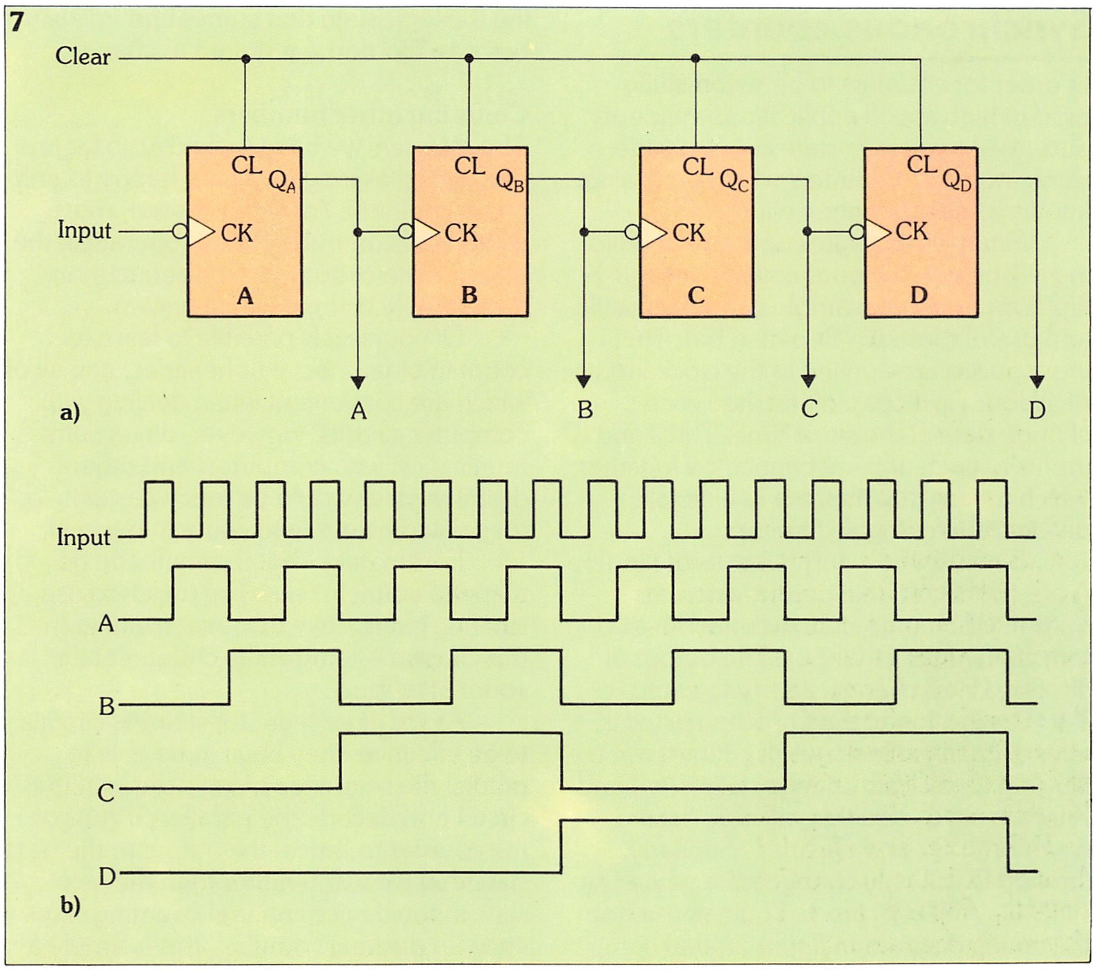
An asynchronous binary counter

Now that we are familiar with the basic varieties of flip-flops, and the ways that they are used in parallel and shift registers, let's move on to another important sequential structural unit, the **counter**. There are various types of counter, however, all can be regarded as serial registers with one input and, usually, a parallel output from each flip-flop in the register. As its name suggests, a counter counts the pulses which arrive at its inputs. The total number of input pulses is stored in the flip-flops and if parallel outputs are provided, the total can be read out as a binary coded number. *Figure 7a* shows how four T-type flip-flops are used to make a 4-bit binary counter. Every time the input signal goes from 1 to 0, the counter increases the 4-bit binary number by 1. When the count reaches 15 (1111), the cycle starts again from zero (0000). The counter can, however, be cleared (i.e. set to zero) at any time, by means of a common asynchronous clear input which goes to all the flip-flops.

The secret of this counter is that the true output (Q) of each flip-flop – except the last one – is connected to the T input of the flip-flop on the right. So, each time one of these outputs goes from 1 to 0, the next flip-flop toggles to the opposite state – as you can see in the timing diagram in *figure 7b*.

If you count the number of input pulses, you will see that it takes sixteen 1-to-0 input transitions to cause one 1-to-0 change at the Q output: this proves that this really is a circuit that can count up to 16 (i.e. from 0 to 15). The parallel outputs Q_A, Q_B, Q_C and Q_D, produce a 4-bit binary number between 0 and 15, with Q_A being the least significant bit. Q_A represents units of one, Q_B units of two, Q_C units of four, and Q_D – the most significant bit – adds up units of eight. We can see that $1 + 2 + 4 + 8 = 15$.

Take another look at the timing diagram and you will notice that it takes two input pulses to make the output of flip-flop A go from 0 to 1 and back to 0; two pulses from Q_A to force Q_B from 0 to 1 and back again, and so on down the chain.



7. (a) Four T-type flip-flops being used to make a 4-bit binary counter; **(b)** timing diagram.

In fact, each flip-flop acts as a divide-by-two circuit, and four flip-flops in a chain like this make a divide-by-16 circuit, because the pulse frequency at the Q_D output is one sixteenth the frequency of the input.

If there were five flip-flops in the chain, what would the output frequency be? Since each flip-flop divides by two, the answer is half of one sixteenth, i.e. one thirty-secondth of the input frequency. The division ratio of any chain of flip-flops is:

$1/2^n$

where n is the number of flip-flops in the chain.

This is also the highest number that the chain can count to. Four flip-flops count to 16 ($2 \times 2 \times 2 \times 2$ or 2^4), five flip-flops count to 32 (2^5), and so on.

The chain of flip-flops in *figure 8* can be referred to either as a divide-by-16 counter or as a modulo-16 counter (modulus refers to the maximum number a counter can count to which is, of course, the same as the number it divides by).

Counters of this type are also known as **ripple counters** because of the way the output transitions ripple down the chain.

Why asychronous?

The transitions in the ripple counter we have just discussed are not synchronised. In fact, even the changes shown in the timing diagram as being simultaneous are actually slightly separated in time. It takes a finite amount of time for an input transition to work its way through a flip-flop circuit to cause the output to change. This is known as a propagation delay.

One effect of this is that there will be times when the binary count from the parallel output is incorrect. However, this will not matter if only the last output in the chain is being used (e.g. for frequency division or as part of a divider chain), but will cause problems if another circuit is looking for true zero from the parallel outputs. Applications requiring strict accuracy need another type of counter.

Synchronous counters

In order for counters to be successfully used in high speed applications their outputs need to change state at exactly the same moment. Counters which do this are known as **synchronous** counters.

Figure 9, illustrates one way of building a 4-bit synchronous counter using J-K flip-flops (a good example of how versatile and useful these flip-flops can be). The input pulses are applied to the clock inputs of all four flip-flops, so that they each change state at the same time. The J and K inputs of each unit are connected together, which means that they act as a simple divide-by-two, toggle flip-flop.

Because the J and K inputs of flip-flop A are connected to a permanent 1, its output will change state with each 1-to-0 transition at the CK input. The output of flip-flop B will change state with each 1-to-0 transition at the Q_A output (that is, after every two clock pulses). However, flip-flop C will only change state if its J and K inputs are 1, and this will only occur when both Q_A and Q_B are 1. Similarly, flip-flop D will only change state when Q_A AND Q_B AND Q_C are 1. You can see from the timing diagram in *figure 7b* that combinations of outputs can only occur at certain times, and by detecting them, the AND gates steer each flip-flop into the next correct state.

These steering gates are a form of decoder, similar to those seen in earlier chapters. In this case, steering the flip-flops is a matter of recognising and then decoding the logic state that comes immediately before a flip-flop is required to change.

Counting other numbers

The counters we have looked at so far are very useful as long as we are happy to only count in binary. As we have seen, most digital and computer circuits operate in the binary system, but people operate more comfortably in the decimal system.

Of course it is possible to learn to count in binary, octal or hexadecimal, all of which are convenient when dealing with computer circuits. However, direct communication with computers and other digital devices would be much easier if *they* could be made to count in *decimal.*

Fortunately, digital circuits *can* be made to count in tens. The trick is to use steering gates – like those seen earlier in this chapter – to reset the counter at the count after nine.

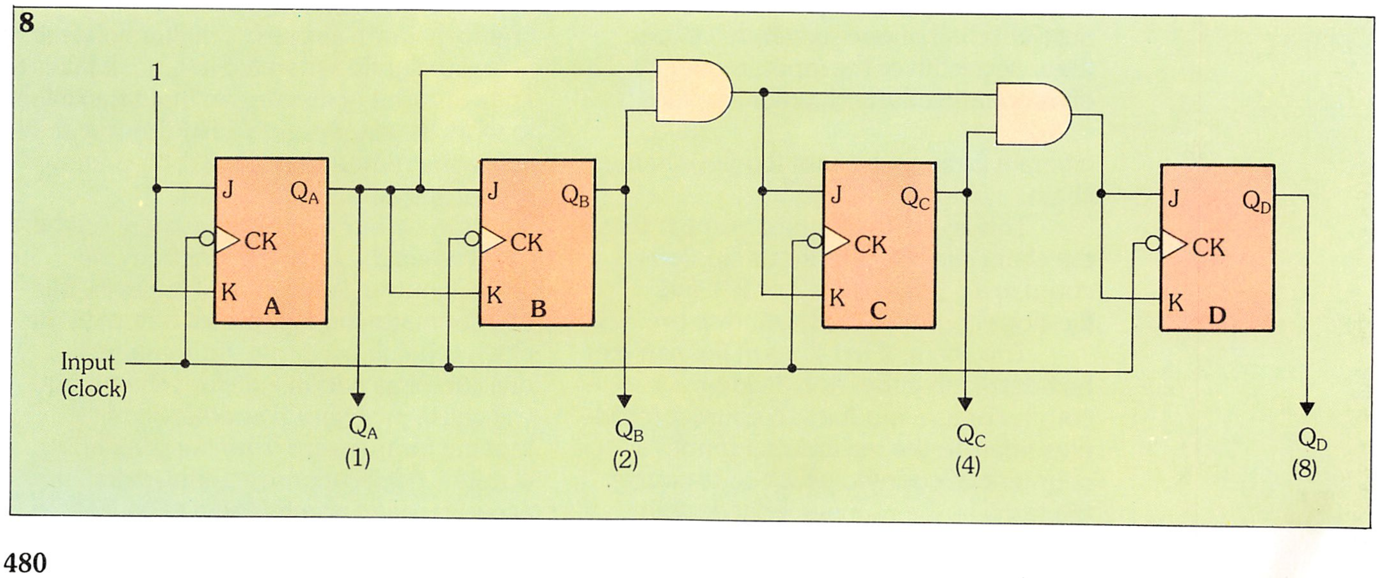
As we have seen, for steering circuits to be effective, they have to be able to predict the sequence of logic states in the circuit and decode the parallel outputs of the counter to 'force' the output to the next state that *we* want, rather than the next state in the devices natural counting sequence. In decimal counting, this is simply a matter of cycling back to zero after the count of nine.

Figure 9 shows how to make a counter which can count from 0 to 9 and back to 0, using binary code. This is a decade (decimal) counter, otherwise called a BCD counter, or a modulo-10 counter.

(continued in part 16)

8. How to build a 4-bit synchronous counter using J-K flip-flops.

8


1
J
Q_A
CK
K
A
J
Q_B
CK
K
B
J
Q_C
CK
K
C
J
Q_D
CK
K
D
Input (clock)
Q_A (1)
Q_B (2)
Q_C (4)
Q_D (8)